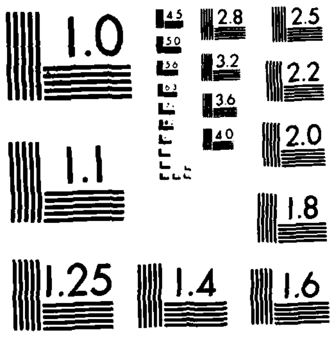
AD-A153 624 INTERNET PROTOCOL IMPLEMENTATION GUIDE(U) SRI INTERNATIONAL MENLO PARK CA NETWORK INFORMATION CENTER AUG 82 DCA200-83-C-0025 1/2

UNCLASSIFIED F/G 17/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

1

INTERNET PROTOCOL IMPLEMENTATION GUIDE

August 1982

Network Information Center
SRI International
Menlo Park, CA 94025
(NIC@NIC)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

A153-62

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION U	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release. Distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Network Information Center SRI International — 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Defense Data Network-PMO
6c. ADDRESS *(City, State, and ZIP Code)* Menlo Park, CA 94025	7b. ADDRESS *(City, State, and ZIP Code)* Washington, DC
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DCA200-83-C-0025
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Internet Protocol Implementation Guide

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
		820800	151

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Internet protocols; Interactive computers; Packet communication; Electronic message handling; Implementation strategy; Window strategy; Internetwork

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This document provides summary and tutorial information on research and development carried out by the DoD on the interconnection and use of packet communication networks. Topics covered include TCP-IP, fault isolation, and gateway connections between dissimilar networks. Guidelines are provided for implementing the Internetwork protocols, along with background papers on the Internetwork protocols and protocols in general.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)* — 22c. OFFICE SYMBOL

E. Redfield EJ223
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025

INTERNET PROTOCOL IMPLEMENTATION GUIDE

August 1982

Network Information Center
SRI International
Menlo Park, CA 94025
(NIC@NIC)

TABLE OF CONTENTS

Note: page numbers listed in the Table of Contents refer to the page numbers in parentheses. Other paging pertains only to the paper in which it occurs. This paging has been left intact so that page references within the text of a given item that refer to the origianl paging will still have meaning.

THE DEPARTMENT OF DEFENSE INTERNET PROTOCOLS

The enclosed package of material provides summary documentation and tutorial information on research and development carried out by the Defense Advanced Research Projects Agency (DARPA), the Defense Communications Agency (DCA), and other parts of the Department of Defense, in the interconnection and use of packet communication networks.

This material reflects the results of several years of development and experimentation with a layered hierarchy of communication protocols and application software designed to support resource sharing, remote interactive computing, and distributed computing services such as electronic message handling.

The protocols have been tested in tactical applications such as fire control and tactical situation reporting, using a mobile packet radio network developed by DARPA and utilizing computing resources on the ARPANET. Logistics applications and internet electronic message handling are in regular use in military testbeds supported by the internet system. The ARPANET network of 300 hosts and approximately 100 packet switches is transitioning to the internet protocol hierarchy, a process which should be completed early in calendar year 1983. In addition, the National Science Foundation is sponsoring a Computer Science Network (CSNET) system which uses the internet protocols on the ARPANET and public Telenet systems.

The protocols are also in regular use across several packet satellite systems using INTELSAT IVA over the Atlantic (SATNET) and the domestic WESTAR (Wide-Band Net/EISN) as well as the Naval FLTSATCOM (MATNET). Local broadband and coaxial cable networks as well as fiber optic nets have been integrated into the system, and the internet protocols have been shown to be efficient for both intranet and internet applications.

The Department of Defense has adopted the Internet Protocol (IP) and the Transmission Control Protocol (TCP) as standards for use in packet networking. The remaining protocols in the hierarchy such as File Transfer Protocol and Terminal/Host Protocol (TELNET) are in regular use in the DARPA experimental Internet System and will be used on an interim basis in operational DoD networks such as the Defense Data Network until formal standards are established.

VINTON G. CERF
Principal Scientist
Information Processing Techniques Office
Defense Advanced Research Projects Agency

THE UNDER SECRETARY OF DEFENSE

WASHINGTON, D.C. 20301

RESEARCH AND ENGINEERING

2 3 MAR 1982

DCA DISTRIBU	
600	
104	
200	X
900	
1100	

Action "A
INFO "I

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
CHAIRMAN OF THE JOINT CHIEFS OF STAFF
DIRECTORS OF THE DEFENSE AGENCIES

SUBJECT: DoD Policy on Standardization of Host-to-Host Protocols for Data Communications Networks

Reference:
(a) USDR&E Memo, "Host-to-Host Protocols for Data Communications Networks," 23 Dec 78
(b) DoD Standard Transmission Control Protocol Specification, Jan 80
(c) DoD Standard Internet Protocol Specification, Jan 80
(d) DoD Directive 4120.3, "Department of Defense Standardization Program," 6 June 73
(e) DoDI 4120.20, "Development and use of Non-Government Specifications and Standards," 28 Dec 76

1. The purpose of this memorandum is to clarify DoD policy concerning standardization of host-to-host protocols for data communications networks.

2. The policy cited in reference (a) is reaffirmed, namely: (1) the use of DoD standard host-to-host protocols (Transmission Control Protocol (TCP) and Internet Protocol (IP), references (b) and (c)) is mandatory for all DoD packet-oriented data networks which have a potential for host-to-host connectivity across network or subnetwork boundaries; (2) the Director, Defense Communications Agency, is designated as the Executive Agent for computer communications protocols; and (3) case-by-case exceptions will be granted by the Executive Agent only for networks shown to have no future requirements for interoperability.

3. Reference (a) is not intended to replace the normal DoD standardization procedures established by DoDD 4120.3 (reference (d)). Rather, the Executive Agent function is intended to place increased emphasis and initiative on the important and currently volatile technology of data communications protocol standardization. New standards and modifications to existing standards will be submitted by the Executive Agent to the Defense Department components for ratification and dissemination in accordance with the provisions of reference (d).

4. DoDI 4120.20 (reference (e)) also continues to apply to protocol standards. Thus, it is desired that nongovernment protocol standards be adopted and used in lieu of the development and promulgation of new

documents. Military requirements for interoperability, security, reliability and survivability are sufficiently pressing to have justified the development and adoption of TCP and IP in the absence of satisfactory nongovernment protocol standards. In the future, the Executive Agent will determine whenever unique military requirements justify the development and adoption of unique DoD protocol standards after making every effort to use prevailing nongovernment standards. Moreover, the Executive Agent will make every effort to inject DoD requirements into the nongovernment standard development process through participation in voluntary standards forums and through coordination with other U.S. Government members of such forums. This influence should be exerted with the objectives of both avoiding the need to develop and adopt unique DoD standards and enabling eventual replacement of unique DoD standards with functionally equivalent nongovernment standards.

[illegible]

Richard D. DeLauer

WINDOW AND ACKNOWLEDGEMENT STRATEGY IN TCP

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. INTRODUCTION

This document describes implementation strategies to deal with two mechanisms in TCP, the window and the acknowledgement. These mechanisms are described in the specification document, but it is possible, while complying with the specification, to produce implementations which yield very bad performance. Happily, the pitfalls possible in window and acknowledgement strategies are very easy to avoid.

It is a much more difficult exercise to verify the performance of a specification than the correctness. Certainly, we have less experience in this area, and we certainly lack any useful formal technique. Nonetheless, it is important to attempt a specification in this area, because different implementors might otherwise choose superficially reasonable algorithms which interact poorly with each other. This document presents a particular set of algorithms which have received testing in the field, and which appear to work properly with each other. With more experience, these algorithms may become part of the formal specification: until such time their use is recommended.

2. THE MECHANISMS

The acknowledgement mechanism is at the heart of TCP. Very simply, when data arrives at the recipient, the protocol requires that it send back an acknowledgement of this data. The protocol specifies that the bytes of data are sequentially numbered, so that the recipient can acknowledge data by naming the highest numbered byte of data it has received, which also acknowledges the previous bytes (actually, it identifies the first byte of data which it has not yet received, but this is a small detail). The protocol contains only a general assertion that data should be acknowledged promptly, but gives no more specific indication as to how quickly an acknowledgement must be sent, or how much data should be acknowledged in each separate acknowledgement.

The window mechanism is a flow control tool. Whenever appropriate, the recipient of data returns to the sender a number, which is (more or less) the size of the buffer which the receiver currently has available for additional data. This number of bytes, called the window, is the maximum which the sender is permitted to transmit until the receiver returns some additional window. Sometimes, the receiver will have no buffer space available, and will return a window value of zero. Under these circumstances,the protocol requires the sender to send a small segment to the receiver now and then, to see if more data is accepted. If the window remains closed at zero for some substantial period, and the sender can obtain no response from the receiver, the protocol requires the sender to conclude that the receiver has failed, and to close the connection. Again, there is very little performance information in the specification, describing under what circumstances the window should be increased, and how the sender should respond to such revised information.

A bad implementation of the window algorithm can lead to extremely poor performance overall. The degradations which occur in throughput and CPU utilizations can easily be several factors of ten, not just a fractional increase. This particular phenomenon is specific enough that it has been given the name of Silly Window Syndrome, or SWS. Happily SWS is easy to avoid if a few simple rules are observed. The most important function of this memo is to describe SWS, so that implementors will understand the general nature of the problem, and to describe algorithms which will prevent its occurrence. This document also describes performance enhancing algorithms which relate to acknowledgement, and discusses the way acknowledgement and window algorithms interact as part of SWS.

3. SILLY WINDOW SYNDROME

In order to understand SWS, we must first define two new terms. Superficially, the window mechanism is very simple: there is a number, called "the window", which is returned from the receiver to the sender. However, we must have a more detailed way of talking about the meaning of this number. The receiver of data computes a value which we will call the "offered window". In a simple case, the offered window corresponds to the amount of buffer space available in the receiver. This correspondence is not necessarily exact, but is a suitable model for the discussion to follow. It is the offered window which is actually transmitted back from the receiver to the sender. The sender uses the offered window to compute a different value, the "usable window", which is the offered window minus the amount of outstanding unacknowledged data. The usable window is less than or equal to the offered window, and can be much smaller.

Consider the following simple example. The receiver initially provides an offered window of 1,000. The sender uses up this window by sending five segments of 200 bytes each. The receiver, on processing the first of these segments, returns an acknowledgement which also contains an updated window value. Let us assume that the receiver of the data has removed the first 200 bytes from the buffer, so that the receiver once again has 1,000 bytes of available buffer. Therefore, the receiver would return, as before, an offered window of 1,000 bytes. The sender, on receipt of this first acknowledgement, now computes the additional number of bytes which may be sent. In fact, of the 1,000 bytes which the recipient is prepared to receive at this time, 800 are already in transit, having been sent in response to the previous offered window. In this case, the usable window is only 200 bytes.

Let us now consider how SWS arises. To continue the previous example, assume that at some point, when the sender computes a useable window of 200 bytes, it has only 50 bytes to send until it reaches a "push" point. It thus sends 50 bytes in one segment, and 150 bytes in the next segment. Sometime later, this 50-byte segment will arrive at the recipient, which will process and remove the 50 bytes and once again return an offered window of 1,000 bytes. However, the sender will now compute that there are 950 bytes in transit in the network, so that the useable window is now only 50 bytes. Thus, the sender will once again send a 50 byte segment, even though there is no longer a natural boundary to force it.

In fact, whenever the acknowledgement of a small segment comes back, the useable window associated with that acknowledgement will cause another segment of the same small size to be sent, until some abnormality breaks the pattern. It is easy to see how small segments arise, because natural boundaries in the data occasionally cause the sender to take a computed useable window and divide it up between two segments. Once that division has occurred, there is no natural way for those useable window allocations to be recombined; thus the breaking up of the useable window into small pieces will persist.

Thus, SWS is a degeneration in the throughput which develops over time, during a long data transfer. If the sender ever stops, as for example when it runs out of data to send, the receiver will eventually acknowledge all the outstanding data, so that the useable window computed by the sender will equal the full offered window of the receiver. At this point the situation will have healed, and further data transmission over the link will occur efficiently. However, in large file transfers, which occur without interruption, SWS can cause appalling performance. The network between the sender and the receiver becomes clogged with many small segments, and an equal number of acknowledgements, which in turn causes lost segments, which triggers massive retransmission. Bad cases of SWS have been seen in which the average segment size was one-tenth of the size the sender and receiver were prepared to deal with, and the average number of retransmission per successful segments sent was five.

Happily, SWS is trivial to avoid. The following sections describe two algorithms, one executed by the sender, and one by the receiver, which appear to eliminate SWS completely. Actually, either algorithm by itself is sufficient to prevent SWS, and thus protect a host from a foreign implementation which has failed to deal properly with this problem. The two algorithms taken together produce an additional reduction in CPU consumption, observed in practice to be as high as a factor of four.

4. IMPROVED WINDOW ALGORITHMS

The receiver of data can take a very simple step to eliminate SWS. When it disposes of a small amount of data, it can artificially reduce the offered window in subsequent acknowledgements, so that the useable window computed by the sender does not permit the sending of any further data. At some later time, when the receiver has processed a substantially larger amount of incoming data, the artificial limitation on the offered window can be removed all at once, so that the sender computes a sudden large jump rather than a sequence of small jumps in the useable window.

At this level, the algorithm is quite simple, but in order to determine exactly when the window should be opened up again, it is necessary to look at some of the other details of the implementation. Depending on whether the window is held artificially closed for a short or long time, two problems will develop. The one we have already discussed -- never closing the window artificially -- will lead to SWS. On the other hand, if the window is only opened infrequently, the pipeline of data in the network between the sender and the receiver may have emptied out while the sender was being held off, so that a delay is introduced before additional data arrives from the sender. This delay does reduce throughput, but it does not consume network resources or CPU resources in the process, as does SWS. Thus, it is in this direction that one ought to overcompensate.

For a simple implementation, a rule of thumb that seems to work in practice is to artificially reduce the offered window until the reduction constitutes one half of the available space, at which point increase the window to advertise the entire space again. In any event, one ought to make the chunk by which the window is opened at least permit one reasonably large segment. (If the receiver is so short of buffers that it can never advertise a large enough buffer to permit at least one large segment, it is hopeless to expect any sort of high throughput.)

There is an algorithm that the sender can use to achieve the same effect described above: a very simple and elegant rule first described by Michael Greenwald at MIT. The sender of the data uses the offered window to compute a useable window, and then compares the useable window to the offered window, and refrains from sending anything if the ratio of useable to offered is less than a certain fraction. Clearly, if the computed useable window is small compared to the offered window, this means that a substantial amount of previously sent information is still in the pipeline from the sender to the receiver, which in turn means that the sender can count on being granted a larger useable window in the future. Until the useable window reaches a certain amount, the sender should simply refuse to send anything.

Simple experiments suggest that the exact value of the ratio is not very important, but that a value of about 25 percent is sufficient to avoid SWS and achieve reasonable throughput, even for machines with a small offered window. An additional enhancement which might help throughput would be to attempt to hold off sending until one can send a maximum size segment. Another enhancement would be to send anyway, even if the ratio is small, if the useable window is sufficient to hold the data available up to the next "push point".

This algorithm at the sender end is very simple. Notice that it is not necessary to set a timer to protect against protocol lockup when postponing the send operation. Further acknowledgements, as they arrive, will inevitably cnange the ratio of offered to useable window. (To see this, note that when all the data in the catanet pipeline has arrived at the receiver, the resulting acknowledgement must yield an offered window and useable window that equal each other.) If the expected acknowledgements do not arrive, the retransmission mechanism will come into play to assure that something finally happens. Thus, to add this algorithm to an existing TCP implementation usually requires one line of code. As part of the send algorithm it is already necessary to compute the useable window from the offered window. It is a simple matter to add a line of code which, if the ratio is less than a certain percent, sets the useable window to zero. The results of SWS are so devastating that no sender should be without this simple piece of insurance.

5. IMPROVED ACKNOWLEDGEMENT ALGORITHMS

In the beginning of this paper, an overly simplistic implementation of TCP was described, which led to SWS. One of the characteristics of this implementation was that the recipient of data sent a separate acknowledgement for every segment that it received. This compulsive acknowledgement was one of the causes of SWS, because each acknowledgement provided some new useable window, but even if one of the algorithms described above is used to eliminate SWS, overly frequent acknowledgement still has a substantial problem, which is that it greatly increases the processing time at the sender's end. Measurement of TCP implementations, especially on large operating systems, indicate that most of the overhead of dealing with a segment is not in the processing at the TCP or IP level, but simply in the scheduling of the handler which is required to deal with the segment. A steady dribble of acknowledgements causes a high overhead in scheduling, with very little to show for it. This waste is to be avoided if possible.

There are two reasons for prompt acknowledgement. One is to prevent retransmission. We will discuss later how to determine whether unnecessary retransmission is occurring. The other reason one acknowledges promptly is to permit further data to be sent. However, the previous section makes quite clear that it is not always desirable to send a little bit of data, even though the receiver may have room for it. Therefore, one can state a general rule that under normal operation, the receiver of data need not, and for efficiency reasons should not, acknowledge the data unless either the acknowledgement is intended to produce an increased useable window, is necessary in order to prevent retransmission or is being sent as part of a reverse direction segment being sent for some other reason. We will consider an algorithm to achieve these goals.

Only the recipient of the data can control the generation of acknowledgements. Once an acknowledgement has been sent from the receiver back to the sender, the sender must process it. Although the extra overhead is incurred at the sender's end, it is entirely under the receiver's control. Therefore, we must now describe an algorithm which occurs at the receiver's end. Obviously, the algorithm must have the following general form; sometimes the receiver of data, upon processing a segment, decides not to send an acknowledgement now, but to postpone the acknowledgement until some time in the future, perhaps by setting a timer. The peril of this approach is that on many large operating systems it is extremely costly to respond to a timer event, almost as costly as to respond to an incoming segment. Clearly, if the receiver of the data, in order to avoid extra overhead at the sender end, spends a great deal of time responding to timer interrupts, no overall benefit has been achieved, for efficiency at the sender end is achieved by great thrashing at the receiver end. We must find an algorithm that avoids both of these perils.

The following scheme seems a good compromise. The receiver of data will refrain from sending an acknowledgement under certain circumstances, in which case it must set a timer which will cause the acknowledgement to be sent later. However, the receiver should do this only where it is a reasonable guess that some other event will intervene and prevent the necessity of the timer interrupt. The most obvious event on which to depend is the arrival of another segment. So, if a segment arrives, postpone sending an acknowledgement if both of the following conditions hold. First, the push bit is not set in the segment, since it is a reasonable assumption that there is more data coming in a subsequent segment. Second, there is no revised window information to be sent back.

This algorithm will insure that the timer, although set, is seldom used. The interval of the timer is related to the expected inter- segment delay, which is in turn a function of the particular network through which the data is flowing. For the ARPANET, a reasonable interval seems to be 200 to 300 milliseconds. Appendix A describes an adaptive algorithm for measuring this delay.

The section on improved window algorithms described both a receiver algorithm and a sender algorithm, and suggested that both should be used. The reason for this is now clear. While the sender algorithm is extremely simple, and useful as insurance, the receiver algorithm is required in order that this improved acknowledgement strategy work. If the receipt of every segment causes a new window value to be returned, then of necessity an acknowledgement will be sent for every data segment. When, according to the strategy of the previous section, the receiver determines to artificially reduce the offered window, that is precisely the circumstance under which an acknowledgement need not be sent. When the receiver window algorithm and the receiver acknowledgement algorithm are used together, it will be seen that sending an acknowledgement will be triggered by one of the following events. First, a push bit has been received. Second, a temporary pause in the data stream is detected. Third, the offered window has been artificially reduced to one-half its actual value.

In the beginning of this section, it was pointed out that there are two reasons why one must acknowledge data. Our consideration at this point has been concerned only with the first, that an acknowledgement must be returned as part of triggering the sending of new data. It is also necessary to acknowledge whenever the failure to do so would trigger retransmission by the sender. Since the retransmission interval is selected by the sender, the receiver of the data cannot make a precise determination of when the acknowledgement must be sent. However, there is a rough rule the sender can use to avoid retransmission, provided that the receiver is reasonably well behaved.

We will assume that sender of the data uses the optional algorithm described in the TCP specification, in which the roundtrip delay is measured using an exponential decay smoothing alg rithm. Retransmission of a segment occurs if the measured delay for that segment exceeds the smoothed average by some factor. To see how retransmission might be triggered, one must consider the pattern of segment arrivals at the receiver.

The goal of our strategy was that the sender should send off a number of segments in close sequence, and receive one acknowledgement for the whole burst. The acknowledgement will be generated by the receiver at the time that the last segment in the burst arrives at the receiver. (To ensure the prompt return of the acknowledgement, the sender could turn on the "push" bit in the last segment of the burst.) The delay observed at the sender between the initial transmission of a segment and the receipt of the acknowledgement will include both the network transit time, plus the holding time at the receiver. The holding time will be greatest for the first segments in the burst, and smallest for the last segments in the burst. Thus, the smoothing algorithm will measure a delay which is roughly proportional to the average roundtrip delay for all the segments in the burst.

Problems will arise if the average delay is substantially smaller than the maximum delay and the smoothing algorithm used has a very small threshold for triggering retransmission. The widest variation between average and maximum delay will occur when network transit time is negligible, and all delay is processing time. In this case, the maximum will be twice the average (by simple algebra) so the threshold that controls retransmission should be somewhat more than a factor of two.

In practice, retransmission of the first segments of a burst has not been a problem because the delay measured consists of the network roundtrip delay, as well as the delay due to withholding the acknowledgement, and the roundtrip tends to dominate except in very low roundtrip time situations (such as when sending to one's self for test purposes). This low roundtrip situation can be covered very simply by including a minimum value below which the roundtrip estimate is not permitted to drop.

In our experiments with this algorithm, retransmission due to faulty calculation of the roundtrip delay occurred only once, when the parameters of the exponential smoothing algorithm had been misadjusted so that they were only taking into account the last two or three segments sent. Clearly, this will cause trouble since the last two or three segments of any burst are the ones whose holding time at the receiver is minimal, so the resulting total estimate was much lower than appropriate. Once the parameters of the algorithm had been adjusted so that the number of segments taken int account was approximately twice the number of segments in a burst of average size, with a threshold factor of 1.5, no further retransmission has ever been identified due to this problem, including when sending to ourself and when sending over high delay nets.

6. CONSERVATIVE VS. OPTIMISTIC WINDOWS

According to the TCP specification, the offered window is presumed to have some relationship to the amount of data which the receiver is actually prepared to receive. However, it is not necessarily an exact correspondence. We will use the term "conservative window" to describe the case where the offered window is precisely no larger than the actual buffering available. The drawback to conservative window algorithms is that they can produce very low throughput in long delay situations. It is easy to see that the maximum input of a conservative window algorithm is one bufferfull every roundtrip delay in the net, since the next bufferfull cannot be launched until the updated window/acknowledgement information from the previous transmission has made the roundtrip.

In certain cases, it may be possible to increase the overall throughput of the transmission by increasing the offered window over the actual buffer available at the receiver. Such a strategy we will call an "optimistic window" strategy. The optimistic strategy works if the network delivers the data to the recipient sufficiently slowly that it can process the data fast enough to keep the buffer from verflowing. If the receiver is faster than the sender, ne could, with luck, permit an infinitely optimistic window, in which the sender is simply permitted to send full-speed. If the sender is faster than the receiver, however, and the window is to optimistic, then some segments will cause a buffer verflow, and will be discarded. Therefore, the correct strategy to implement an optimistic window is to increase the window size until segments start to be lost. This only works if it is possible to detect that the segment has been lost.

In some cases, it is easy to do, because the segment is partially processed inside the receiving host before it is thrown away. In other cases, overflows may actually cause the network interface to be clogged, which will cause the segments to be lost elsewhere in the net. It is inadvisable to attempt an optimistic window strategy unless one is certain that the algorithm can detect the resulting lost segments. However, the increase in throughput which is possible from optimistic windows is quite substantial. Any systems with small buffer space should seriously consider the merit of optimistic windows.

The selection of an appropriate window algorithm is actually more complicated than even the above discussion suggests. The following considerations are not presented with the intention that they be incorporated in current implementations of TCP, but as background for the sophisticated designer who is attempting to understand how his TCP will respond to a variety of networks, with different speed and delay characteristics. The particular pattern of windows and acknowledgements sent from receiver to sender influences two characteristics of the data being sent. First, they control the average data rate. Clearly, the average rate of the sender cannot exceed the average rate of the receiver, or long-term buffer overflow will occur. Second, they influence the burstiness of the data coming from the sender. Burstiness has both advantages and disadvantages. The advantage of burstiness is that it reduces the CPU processing necessary to send the data. This follows from the observed fact, especially on large machines, that most of the cost of sending a segment is not the TCP or IP processing, but the scheduling overhead of getting started.

On the other hand, the disadvantage of burstiness is that it may cause buffers to overflow, either in the eventual recipient, which was discussed above, or in an intermediate gateway, a problem ignored in this paper. The algorithms described above attempts to strike a balance between excessive burstiness, which in the extreme cases can cause delays because a burst is not requested soon enough, and excessive fragmentation of the data stream into small segments, which we identified as Silly Window Syndrome.

Under conditions of extreme delay in the network, none of the algorithms described above will achieve adequate throughput. Conservative window algorithms have a predictable throughput limit, which is one windowfull per roundtrip delay. Attempts to solve this by optimistic window strategies .nay cause buffer overflows due to the bursty nature of the arriving data. A very sophisticated way to solve this is for the receiver, having measured by some means the roundtrip delay and intersegment arrival rate of the actual connection, to open his window, not in one optimistic increment of gigantic proportion, but in a number of smaller optimistic increments, which have been carefully spaced using a timer so that the resulting smaller bursts which arrive are each sufficiently small to fit into the existing buffers. One could visualize this as a number of requests flowing backwards through the net which trigger in return a number of bursts which flow back spaced evenly from the sender to the receiver. The overall result is that the receiver uses the window mechanism to control the burstiness of the arrivals, and the average rate.

To my knowledge, no such strategy has been implemented in any TCP. First, we do not normally have delays high enough to require this kind of treatment. Second, the strategy described above is probably not stable unless it is very carefully balanced. Just as buses on a single bus route tend to bunch up, bursts which start out equally spaced could well end up piling into each other, and forming the single large burst which the receiver was hoping to avoid. It is important to understand this extreme case, however, in order to understand the limits beyond which TCP, as normally implemented, with either conservative or simple optimistic windows can be expected to deliver throughput which is a reasonable percentage of the actual network capacity.

7. CONCLUSIONS

This paper describes three simple algorithms for performance enhancement in TCP, one at the sender end and two at the receiver. The sender algorithm is to refrain from sending if the useable window is smaller than 25 percent of the offered window. The receiver algorithms are first, to artificially reduce the offered window when processing new data if the resulting reduction does not represent more than some fraction, say 50 percent, of the actual space available, and second, to refrain from sending an acknowledgment at all if two simple conditions hold.

Either of these algorithms will prevent the worst aspects of Silly Window Syndrome, and when these algorithms are used together, they will produce substantial improvement in CPU utilization, by eliminating the process of excess acknowledgements.

Preliminary experiments with these algorithms suggest that they work, and work very well. Both the sender and receiver algorithms have been shown to eliminate SWS, even when talking to fairly silly algorithms at the other end. The Multics mailer, in particular, had suffered substantial attacks of SWS while sending large mail to a number of hosts. We believe that implementation of the sender side algorithm has eliminated every known case of SWS detected in our mailer. Implementation of the receiver side algorithm produced substantial improvements of CPU time when Multics was the sending system.

Multics is a typical large operating system, with scheduling costs which are large compared to the actual processing time for protocol handlers. Tests were done sending from Multics to a host which implemented the SWS suppression algorithm, and which could either refrain or not from sending acknowledgements on each segment. As predicted, suppressing the return acknowledgements did not influence the throughput for large data transfer at all, since the throttling effect was elsewhere. However, the CPU time required to process the data at the Multics end was cut by a factor of four (In this experiment, the bursts of data which were being sent were approximately eight segments. Thus, the number of acknowledgements in the two experiments differed by a factor of eight.)

An important consideration in evaluating these algorithms is that they must not cause the protocol implementations to deadlock. All of the recommendations in this document have the characteristic that they suggest one refrain from doing something even though the protocol specification permits one to do it. The possibility exists that if one refrains from doing something now one may never get to do it later, and both ends will halt, even though it would appear superficially that the transaction can continue.

Formally, the idea that things continue to work is referred to as "liveness". One of the defects of ad hoc solutions to performance problems is the possibility that two different approaches will interact to prevent liveness. It is believed that the algorithms described in this paper are always live, and that is one of the reasons why there is a strong advantage in uniform use of this particular proposal, except in cases where it is explicitly demonstrated not to work.

The argument for liveness in these solutions proceeds as follows. First, the sender algorithm can only be stopped by one thing, a refusal of the receiver to acknowledge sent data. As long as the receiver continues to acknowledge data, the ratio of useable window to offered window will approach one, and eventually the sender must continue to send. However, notice that the receiver algorithm we have advocated involves refraining from acknowledging. Therefore, we certainly do have a situation where improper operation of this algorithm can prevent liveness.

What we must show is that the receiver of the data, if it chooses to refrain from acknowledging, will do so only for a short time, and not forever. The design of the algorithm described above was intended to achieve precisely this goal: whenever the receiver of data refrained from sending an acknowledgement it was required to set a timer. The only event that was permitted to clear that timer was the receipt of another segment, which essentially reset the timer, and started it going again. Thus, an acknowledgement will be sent as soon as no data has been received. This has precisely the effect desired: if the data flow appears to be disrupted for any reason, the receiver responds by sending an up-to-date acknowledgement. In fact, the receiver algorithm is designed to be more robust than this, for transmission of an acknowledgment is triggered by two events, either a cessation of data or a reduction in the amount of offered window to 50 percent of the actual value. This is the condition which will normally trigger the transmission of this acknowledgement.

APPENDIX A

DYNAMIC CALCULATION OF ACKNOWLEDGEMENT DELAY

The text suggested that when setting a timer to postpone the sending of an acknowledgement, a fixed interval of 200 to 300 milliseconds would work properly in practice. This has not been verified over a wide variety of network delays, and clearly if there is a very slow net which stretches out the intersegment arrival time, a fixed interval will fail. In a sophisticated TCP, which is expected to adjust dynamically (rather than manually) to changing network conditions, it would be appropriate to measure this interval and respond dynamically.

The following algorithm, which has been relegated to an Appendix, because it has not been tested, seems sensible. Whenever a segment arrives which does not have the push bit on in it, start a timer, which runs until the next segment arrives. Average these interarrival intervals, using an exponential decay smoothing function tuned to take into account perhaps the last ten or twenty segments that have come in. Occasionally, there will be a long interarrival period, even for a segment which does not terminate a piece of data being pushed, perhaps because a window has gone to zero or some glitch in the sender or the network has held up the data. Therefore, examine each interarrival interval, and discard it from the smoothing algorithm if it exceeds the current estimate by some amount, perhaps a ratio of two or four times. By rejecting the larger intersegment arrival intervals, one should obtain a smoothed estimate of the interarrival of segments inside a burst. The number need not be exact, since the timer which triggers acknowledgement can add a fairly generous fudge factor to this without causing trouble with the sender's estimate of the retransmission interval, so long as the fudge factor is constant.

NAME, ADDRESSES, PORTS, AND ROUTES

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. INTRODUCTION

It has been said that the principal function of an operating system is to define a number of different names for the same object, so that it can busy itself keeping track of the relationship between all of the different names. Network protocols seem to have somewhat the same characteristic. In TCP/IP, there are several ways of referring to things. At the human visible interface, there are character string "names" to identify networks, hosts, and services. Host names are translated into network "addresses", 32-bit values that identify the network to which a host is attached, and the location of the host on that net. Service names are translated into a "port identifier", which in TCP is a 16-bit value. Finally, addresses are translated into "routes", which are the sequence of steps a packet must take to reach the specified addresses. Routes show up explicitly in the form of the internet routing options, and also implicitly in the address to route translation tables which all hosts and gateways maintain.

This RFC gives suggestions and guidance for the design of the tables and algorithms necessary to keep track of these various sorts of identifiers inside a host implementation of TCP/IP.

2. THE SCOPE OF THE PROBLEM

One of the first questions one can ask about a naming mechanism is how many names one can expect to encounter. In order to answer this, it is necessary to know something about the expected maximum size of the internet. Currently, the internet is fairly small. It contains no more than 25 active networks, and no more than a few hundred hosts. This makes it possible to install tables which exhaustively list all of these elements. However, any implementation undertaken now should be based on an assumption of a much larger internet. The guidelines currently recommended are an upper limit of about 1,000 networks. If we imagine an average number of 25 hosts per net, this would suggest a maximum number of 25,000 hosts. It is quite unclear whether this host estimate is high or low, but even if it is off by several factors of two, the resulting number is still large enough to suggest that current table management strategies are unacceptable. Some fresh techniques will be required to deal with the internet of the future.

3. NAMES

As the previous section suggests, the internet will eventually have a sufficient number of names that a host cannot have a static table which provides a translation from every name to its associated address. There are several reasons other than sheer size why a host would not wish to have such a table. First, with that many names, we can expect names to be added and deleted at such a rate that an installer might spend all his time just revising the table. Second, most of the names will refer to addresses of machines with which nothing will ever be exchanged. In fact, there may be whole networks with which a particular host will never have any traffic.

To cope with this large and somewhat dynamic environment, the internet is moving from its current position in which a single name table is maintained by the NIC and distributed to all hosts, to a distributed approach in which each network (or group of networks) is responsible for maintaining its own names and providing a "name server" to translate between the names and the addresses in that network. Each host is assumed to store not a complete set of name-address translations, but only a cache of recently used names. When a name is provided by a user for translation to an address, the host will first examine its local cache, and if the name is not found there, will communicate with an appropriate name server to obtain the information, which it may then insert into its cache for future reference.

Unfortunately, the name server mechanism is not totally in place in the internet yet, so for the moment, it is necessary to continue to use the old strategy of maintaining a complete table of all names in every host. Implementors, however, should structure this table in such a way that it is easy to convert later to a name server approach. In particular, a reasonable programming strategy would be to make the name table accessible only through a subroutine interface, rather than by scattering direct references to the table all through the code. In this way, it will be possible, at a later date, to replace the subroutine with one capable of making calls on remote name servers.

A problem which occasionally arises in the ARPANET today is that the information in a local host table is out of date, because a host has moved, and a revision of the host table has not yet been installed from the NIC. In this case, one attempts to connect to a particular host and discovers an unexpected machine at the address obtained from the local table. If a human is directly observing the connection attempt, the error is usually detected immediately. However, for unattended operations such as the sending of queued mail, this sort of problem can lead to a great deal of confusion.

The nameserver scheme will only make this problem worse, if hosts cache locally the address associated with names that have been looked up, because the host has no way of knowing when the address has changed and the cache entry should be removed. To solve this problem, plans are currently under way to define a simple facility by which a host can query a foreign address to determine what name is actually associated with it. SMTP already defines a verification technique based on this approach.

4. ADDRESSES

The IP layer must know something about addresses. In particular, when a datagram is being sent out from a host, the IP layer must decide where to send it on the immediately connected network, based on the internet address. Mechanically, the IP first tests the internet address to see whether the network number of the recipient is the same as the network number of the sender. If so, the packet can be sent directly to the final recipient. If not, the datagram must be sent to a gateway for further forwarding. In this latter case, a second decision must be made, as there may be more than one gateway available on the immediately attached network.

When the internet address format was first specified, 8 bits were reserved to identify the network. Early implementations thus implemented the above algorithm by means of a table with 256 entries, one for each possible net, that specified the gateway of choice for that net, with a special case entry for those nets to which the host was immediately connected. Such tables were sometimes statically filled in, which caused confusion and malfunctions when gateways and networks moved (or crashed).

The current definition of the internet address provides three different options for network numbering, with the goal of allowing a very large number of networks to be part of the internet. Thus, it is no longer possible to imagine having an exhaustive table to select a gateway for any foreign net. Again, current implementations must use a strategy based on a local cache of routing information for addresses currently being used.

The recommended strategy for address to route translation is as follows. When the IP layer receives an outbound datagram for transmission, it extracts the network number from the destination address, and queries its local table to determine whether it knows a suitable gateway to which to send the datagram. If it does, the job is done. (But see RFC 816 on Fault Isolation and Recovery, for recommendations on how to deal with the possible failure of the gateway.) If there is no such entry in the local table, then select any accessible gateway at random, insert that as an entry in the table, and use it to send the packet. Either the guess will be right or wrong. If it is wrong, the gateway to which the packet was sent will return an ICMP redirect message to report that there is a better gateway to reach the net in question. The arrival of this redirect should cause an update of the local table.

The number of entries in the local table should be determined by the maximum number of active connections which this particular host can support at any one time. For a large time sharing system, one might imagine a table with 100 or more entries. For a personal computer being used to support a single user telnet connection, only one address to gateway association need be maintained at once.

The above strategy actually does not completely solve the problem, but only pushes it down one level, where the problem then arises of how a new host, freshly arriving on the internet, finds all of its accessible gateways. Intentionally, this problem is not solved within the internetwork architecture. The reason is that different networks have drastically different strategies for allowing a host to find out about other hosts on its immediate network. Some nets permit a broadcast mechanism. In this case, a host can send out a message and expect an answer back from all of the attached gateways. In other cases, where a particular network is richly provided with tools to support the internet, there may be a special network mechanism which a host can invoke to determine where the gateways are. In other cases, it may be necessary for an installer to manually provide the name of at least one accessible gateway. Once a host has discovered the name of one gateway, it can build up a table of all other available gateways, by keeping track of every gateway that has been reported back to it in an ICMP message.

5. ADVANCED TOPICS IN ADDRESSING AND ROUTING

The preceding discussion describes the mechanism required in a minimal implementation, an implementation intended only to provide operational service access today to the various networks that make up the internet. For any host which will participate in future research, as contrasted with service, some additional features are required. These features will also be helpful for service hosts if they wish to obtain access to some of the more exotic networks which will become part of the internet over the next few years. All implementors are urged to at least provide a structure into which these features could be later integrated.

There are several features, either already a part of the architecture or now under development, which are used to modify or expand the relationships between addresses and routes. The IP source route options allow a host to explicitly direct a datagram through a series of gateways to its foreign host. An alternative form of the ICMP redirect packet has been proposed, which would return information specific to a particular destination host, not a destination net. Finally, additional IP options have been proposed to identify particular routes within the internet that are unacceptable.

The difficulty with implementing these new features is that the mechanisms do not lie entirely within the bounds of IP. All the mechanisms above are designed to apply to a particular connection, so that their use must be specified at the TCP level. Thus, the interface between IP and the layers above it must include mechanisms to allow passing this information back and forth, and TCP (or any other protocol at this level, such as UDP), must be prepared to store this information. The passing of information between IP and TCP is made more complicated by the fact that some of the information, in particular ICMP packets, may arrive at any time. The normal interface envisioned between TCP and IP is one across which packets can be sent or received. The existence of asynchronous ICMP messages implies that there must be an additional channel between the two, unrelated to the actual sending and receiving of data. (In fact, there are many other ICMP messages which arrive asynchronously and which must be passed from IP up to higher layers. See RFC 816, Fault Isolation and Recovery.)

Source routes are already in use in the internet, and many implementations will wish to be able to take advantage of them. The following sorts of usages should be permitted. First, a user, when initiating a TCP connection, should be able to hand a source route into TCP, which in turn must hand the source route to IP with every outgoing datagram. The user might initially obtain the source route by querying a different sort of name server, which would return a source route instead of an address, or the user may have fabricated the source route manually. A TCP which is listening for a connection, rather than attempting to open one, must be prepared to receive a datagram which contains a IP return route, in which case it must remember this return route, and use it as a source route on all returning datagrams.

6. PORTS AND SERVICE IDENTIFIERS

The IP layer of the architecture contains the address information which specifies the destination host to which the datagram is being sent. In fact, datagrams are not intended just for particular hosts, but for particular agents within a host, processes or other entities that are the actual source and sink of the data. IP performs only a very simple dispatching once the datagram has arrived at the target host, it dispatches it to a particular protocol. It is the responsibility of that protocol handler, for example TCP, to finish dispatching the datagram to the particular connection for which it is destined. This next layer of dispatching is done using "port identifiers", which are a part of the header of the higher level protocol, and not the IP layer.

This two-layer dispatching architecture has caused a problem for certain implementations. In particular, some implementations have wished to put the IP layer within the kernel of the operating system, and the TCP layer as a user domain application program. Strict adherence to this partitioning can lead to grave performance problems, for the datagram must first be dispatched from the kernel to a TCP process, which then dispatches the datagram to its final destination process. The overhead of scheduling this dispatch process can severely limit the achievable throughput of the implementation.

As is discussed in RFC 817, Modularity and Efficiency in Protocol Implementations, this particular separation between kernel and user leads to other performance problems, even ignoring the issue of port level dispatching. However, there is an acceptable shortcut which can be taken to move the higher level dispatching function into the IP layer, if this makes the implementation substantially easier.

In principle, every higher level protocol could have a different dispatching algorithm. The reason for this is discussed below. However, for the protocols involved in the service offering being implemented today, TCP and UDP, the dispatching algorithm is exactly the same, and the port field is located in precisely the same place in the header. Therefore, unless one is interested in participating in further protocol research, there is only one higher level dispatch algorithm.

This algorithm takes into account the internet level foreign address, the protocol number, and the local port and foreign port from the higher level protocol header. This algorithm can be implemented as a sort of adjunct to the IP layer implementation, as long as no other higher level protocols are to be implemented. (Actually, the above statement is only partially true, in that the UDP dispatch function is subset of the TCP dispatch function. UDP dispatch depends only protocol number and local port. However, there is an occasion within TCP when this exact same subset comes into play, when a process wishes to listen for a connection from any foreign host. Thus, the range of mechanisms necessary to support TCP dispatch are also sufficient to support precisely the UDP requirement.)

The decision to remove port level dispatching from IP to the higher level protocol has been questioned by some implementors. It has been argued that if all of the address structure were part of the IP layer, then IP could do all of the packet dispatching function within the host, which would lead to a simpler modularity. Three problems were identified with this.

First, not all protocol implementors could agree on the size of the port identifier. TCP selected a fairly short port identifier, 16 bits, to reduce header size. Other protocols being designed, however, wanted a larger port identifier, perhaps 32 bits, so that the port identifier, if properly selected, could be considered probabilistically unique. Thus, constraining the port id to one particular IP level mechanism would prevent certain fruitful lines of research.

Second, ports serve a special function in addition to datagram delivery: certain port numbers are reserved to identify particular services. Thus, TCP port 23 is the remote login service. If ports were implemented at the IP level, then the assignment of well known ports could not be done on a protocol basis, but would have to be done in a centralized manner for all of the IP architecture.

Third, IP was designed with a very simple layering role: IP contained exactly those functions that the gateways must understand. If the port idea had been made a part of the IP layer, it would have suggested that gateways needed to know about ports, which is not the case.

There are, of course, other ways to avoid these problems. In particular, the "well-known port" problem can be solved by devising a second mechanism, distinct from port dispatching, to name well-known ports. Several protocols have settled on the idea of including, in the packet which sets up a connection to a particular service, a more general service descriptor, such as a character string field. These special packets, which are requesting connection to a particular service, are routed on arrival to a special server, sometimes called a "rendezvous server", which examines the service request, selects a random port which is to be used for this instance of the service, and then passes the packet along to the service itself to commence the interaction.

For the internet architecture, this strategy had the serious flaw that it presumed all protocols would fit into the same service paradigm: an initial setup phase, which might contain a certain overhead such as indirect routing through a rendezvous server, followed by the packets of the interaction itself, which would flow directly to the process providing the service.

Unfortunately, not all high level protocols in internet were expected to fit this model. The best example of this is isolated datagram exchange using UDP. The simplest exchange in UDP is one process sending a single datagram to another. Especially on a local net, where the net related overhead is very low, this kind of simple single datagram interchange can be extremely efficient, with very low overhead in the hosts. However, since these individual packets would not be part of an established connection, if IP supported a strategy based on a rendezvous server and service descriptors, every isolated datagram would have to be routed indirectly in the receiving host through the rendezvous server, which would substantially increase the overhead of processing, and every datagram would have to carry the full service request field, which would increase the size of the packet header.

In general, if a network is intended for "virtual circuit service", or things similar to that, then using a special high overhead mechanism for circuit setup makes sense. However, current directions in research are leading away from this class of protocol, so once again the architecture was designed not to preclude alternative protocol structures. The only rational position was that the particular dispatching strategy used should be part of the higher level protocol design, not the IP layer.

This same argument about circuit setup mechanisms also applies to the design of the IP address structure. Many protocols do not transmit a full address field as part of every packet, but rather transmit a short identifier which is created as part of a circuit setup from source to destination. If the full address needs to be carried in only the first packet of a long exchange, then the overhead of carrying a very long address field can easily be justified. Under these circumstances, one can create truly extravagant address fields, which are capable of extending to address almost any conceivable entity. However, this strategy is usable only in a virtual circuit net, where the packets being transmitted are part of a established sequence, otherwise this large extravagant address must be transported on every packet.

Since Internet explicitly rejected this restriction on the architecture, it was necessary to come up with an address field that was compact enough to be sent in every datagram, but general enough to correctly route the datagram through the catanet without a previous setup phase. The IP address of 32 bits is the compromise that results. Clearly it requires a substantial amount of shoehorning to address all of the interesting places in the universe with only 32 bits. On the other hand, had the address field become much bigger, IP would have been susceptible to another criticism, which is that the header had grown unworkably large. Again, the fundamental design decision was that the protocol be designed in such a way that it supported research in new and different sorts of protocol architectures.

There are some limited restrictions imposed by the IP design on the port mechanism selected by the higher level process. In particular, when a packet goes awry somewhere on the internet, the offending packet is returned, along with an error indication, as part of an ICMP packet. An ICMP packet returns only the IP layer, and the next 64 bits of the original datagram. Thus, any higher level protocol which wishes to sort out from which port a particular offending datagram came must make sure that the port information is contained within the first 64 bits of the next level header. This also means, in most cases, that it is possible to imagine, as part of the IP layer, a port dispatch mechanism which works by masking and matching on the first 64 bits of the incoming higher level header.

IP DATAGRAM REASSEMBLY ALGORITHMS

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. INTRODUCTION

One of the mechanisms of IP is fragmentation and reassembly. Under certain circumstances, a datagram originally transmitted as a single unit will arrive at its final destination broken into several fragments. The IP layer at the receiving host must accumulate these fragments until enough have arrived to completely reconstitute the original datagram. The specification document for IP gives a complete description of the reassembly mechanism, and contains several examples. It also provides one possible algorithm for reassembly, based on keeping track of arriving fragments in a vector of bits. This document describes an alternate approach which should prove more suitable in some machines.

A superficial examination of the reassembly process may suggest that it is rather complicated. First, it is necessary to keep track of all the fragments, which suggests a small bookkeeping job. Second, when a new fragment arrives, it may combine with the existing fragments in a number of different ways. It may precisely fill the space between two fragments, or it may overlap with existing fragments, or completely duplicate existing fragments, or partially fill a space between two fragments without abutting either of them. Thus, it might seem that the reassembly process might involve designing a fairly complicated algorithm that tests for a number of different options.

In fact, the process of reassembly is extremely simple. This document describes a way of dealing with reassembly which reduces the bookkeeping problem to a minimum, which requires for storage only one buffer equal in size to the final datagram being reassembled, which can reassemble a datagram from any number of fragments arriving in any order with any possible pattern of overlap and duplication, and which is appropriate for almost any sort of operating system.

The reader should consult the IP specification document to be sure that he is completely familiar with the general concept of reassembly, and the particular header fields and vocabulary used to describe the process.

2. THE ALGORITHM

In order to define this reassembly algorithm, it is necessary to define some terms. A partially reassembled datagram consists of certain sequences of octets that have already arrived, and certain areas still to come. We will refer to these missing areas as "holes". Each hole can be characterized by two numbers, hole.first, the number of the first octet in the hole, and hole.last, the number of the last octet in the hole. This pair of numbers we will call the "hole descriptor", and we will assume that all of the hole descriptors for a particular datagram are gathered together in the "hole descriptor list".

The general form of the algorithm is as follows. When a new fragment of the datagram arrives, it will possibly fill in one or more of the existing holes. We will examine each of the entries in the hole descriptor list to see whether the hole in question is eliminated by this incoming fragment. If so, we will delete that entry from the list. Eventually, a fragment will arrive which eliminates every entry from the list. At this point, the datagram has been completely reassembled and can be passed to higher protocol levels for further processing.

The algorithm will be described in two phases. In the first part, we will show the sequence of steps which are executed when a new fragment arrives, in order to determine whether or not any of the existing holes are filled by the new fragment. In the second part of this description, we will show a ridiculously simple algorithm for management of the hole descriptor list.

3. FRAGMENT PROCESSING ALGORITHM

An arriving fragment can fill any of the existing holes in a number of ways. Most simply, it can completely fill a hole. Alternatively, it may leave some remaining space at either the beginning or the end of an existing hole. Or finally, it can lie in the middle of an existing hole, breaking the hole in half and leaving a smaller, hole at each end. Because of these possibilities, it might seem that a number of tests must be made when a new fragment arrives, leading to a rather complicated algorithm. In fact, if properly expressed, the algorithm can compare each hole to the arriving fragment in only four tests.

We start the algorithm when the earliest fragment of the datagram arrives. We begin by creating an empty data buffer area and putting one entry in its hole descriptor list, the entry which describes the datagram as being completely missing. In this case, hole.first equals zero, and hole.last equals infinity. (Infinity is presumably implemented by a very large integer, greater than 576, of the implementor's choice.) The following eight steps are then used to insert each of the arriving fragments into the buffer area where the complete datagram is being built up. The arriving fragment is described by fragment.first, the first octet of the fragment, and fragment.last, the last octet of the fragment.

1. Select the next hole descriptor from the hole descriptor list. If there are no more entries, go to step eight.

2. If fragment.first is greater than hole.last, go to step one.

3. If fragment.last is less than hole.first, go to step one.

   (If either step two or step three is true, then the newly arrived fragment does not overlap with the hole in any way, so we need pay no further attention to this hole. We return to the beginning of the algorithm where we select the next hole for examination.)

4. Delete the current entry from the hole descriptor list.

   (Since neither step two nor step three was true, the newly arrived fragment does interact with this hole in some way. Therefore, the current descriptor will no longer be valid. We will destroy it, and in the next two steps we will determine whether or not it is necessary to create any new hole descriptors.)

5. If fragment.first is greater than hole.first, then create a new hole descriptor "new_hole" with new_hole.first equal to hole.first, and new_hole.last equal to fragment.first minus one.

   (If the test in step five is true, then the first part of the original hole is not filled by this fragment. We create a new descriptor for this smaller hole.)

6. If fragment.last is less than hole.last and fragment.more fragments is true, then create a new hole descriptor "new_hole", with new_hole.first equal to fragment.last plus one and new_hole.last equal to hole.last.

   (This test is the mirror of step five with one additional feature. Initially, we did not know how long the reassembled datagram would be, and therefore we created a hole reaching from zero to infinity. Eventually, we will receive the last fragment of the datagram. At this point, that hole descriptor which reaches from the last octet of the buffer to infinity can be discarded. The fragment which contains the last fragment indicates this fact by a flag in the internet header called "more fragments". The test of this bit in this statement prevents us from creating a descriptor for the unneeded hole which describes the space from the end of the datagram to infinity.)

7. Go to step one.

8. If the hole descriptor list is now empty, the datagram is now complete. Pass it on to the higher level protocol processor for further handling. Otherwise, return.

4. MANAGING THE HOLE DESCRIPTOR LIST

The main complexity in the eight step algorithm above is not performing the arithmetical tests, but in adding and deleting entries from the hole descriptor list. One could imagine an implementation in which the storage management package was many times more complicated than the rest of the algorithm, since there is no specified upper limit on the number of hole descriptors which will exist for a datagram during reassembly. There is a very simple way to deal with the hole descriptors, however. Just put each hole descriptor in the first octets of the hole itself. Note that by the definition of the reassembly algorithm, the minimum size of a hole is eight octets. To store hole.first and hole.last will presumably require two octets each. An additional two octets will be required to thread together the entries on the hole descriptor list. This leaves at least two more octets to deal with implementation idiosyncrasies.

There is only one obvious pitfall to this storage strategy. One must execute the eight step algorithm above before copying the data from the fragment into the reassembly buffer. If one were to copy the data first, it might smash one or more hole descriptors. Once the algorithm above has been run, any hole descriptors which are about to be smashed have already been rendered obsolete.

5. LOOSE ENDS

Scattering the hole descriptors throughout the reassembly buffer itself requires that they be threaded onto some sort of list so that they can be found. This in turn implies that there must be a pointer to the head of the list. In many cases, this pointer can be stored in some sort of descriptor block which the implementation associates with each reassembly buffer. If no such storage is available, a dirty but effective trick is to store the head of the list in a part of the internet header in the reassembly buffer which is no longer needed. An obvious location is the checksum field.

When the final fragment of the datagram arrives, the packet length field in the internet header should be filled in.

6. OPTIONS

The preceding description made one unacceptable simplification. It assumed that there were no internet options associated with the datagram being reassembled. The difficulty with options is that until one receives the first fragment of the datagram, one cannot tell how big the internet header will be. This is because, while certain options are copied identically into every fragment of a datagram, other options, such as "record route", are put in the first fragment only. (The "first fragment" is the fragment containing octet zero of the original datagram.)

Until one knows how big the internet header is, one does not know where to copy the data from each fragment into the reassembly buffer. If the earliest fragment to arrive happens to be the first fragment, then this is no problem. Otherwise, there are two solutions. First, one can leave space in the reassembly buffer for the maximum possible internet header. In fact, the maximum size is not very large, 64 octets. Alternatively, one car simply gamble that the first fragment will contain no options. If, when the first fragment finally arrives, there are options, one can then shift the data in the buffer a sufficient distance for allow for them. The only peril in copying the data is that one will trash the pointers that thread the hole descriptors together. It is easy to see how to untrash the pointers.

The source and record route options have the interesting feature that, since different fragments can follow different paths, they may arrive with different return routes recorded in different fragments. Normally, this is more information than the receiving Internet module needs. The specified procedure is to take the return route recorded in the first fragment and ignore the other versions.

7. THE COMPLETE ALGORITHM

In addition to the algorithm described above there are two parts to the reassembly process. First, when a fragment arrives, it is necessary to find the reassembly buffer associated with that fragment. This requires some mechanism for searching all the existing reassembly buffers. The correct reassembly buffer is identified by an equality of the following fields: the foreign and local internet address, the protocol ID, and the identification field.

The final part of the algorithm is some sort of timer based mechanism which decrements the time to live field of each partially reassembled datagram, so that incomplete datagrams which have outlived their usefulness can be detected and deleted. One can either create a demon which comes alive once a second and decrements the field in each datagram by one, or one can read the clock when each first fragment arrives, and queue some sort of timer call, using whatever system mechanism is appropriate, to reap the datagram when its time has come.

An implementation of the complete algorithm comprising all these parts was constructed in BCPL as a test. The complete algorithm took less than one and one-half pages of listing, and generated approximately 400 nova machine instructions. That portion of the algorithm actually involved with management of hole descriptors is about 20 lines of code.

The version of the algorithm described here is actually a simplification of the author's original version, thanks to an insightful observation by Elizabeth Martin at MIT.

FAULT ISOLATION AND RECOVERY

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. INTRODUCTION

Occasionally, a network or a gateway will go down, and the sequence of hops which the packet takes from source to destination must change. Fault isolation is that action which hosts and gateways collectively take to determine that something is wrong; fault recovery is the identification and selection of an alternative route which will serve to reconnect the source to the destination. In fact, the gateways perform most of the functions of fault isolation and recovery. There are, however, a few actions which hosts must take if they wish to provide a reasonable level of service. This document describes the portion of fault isolation and recovery which is the responsibility of the host.

2. WHAT GATEWAYS DO

Gateways collectively implement an algorithm which identifies the best route between all pairs of networks. They do this by exchanging packets which contain each gateway's latest opinion about the operational status of its neighbor networks and gateways. Assuming that this algorithm is operating properly, one can expect the gateways to go through a period of confusion immediately after some network or gateway has failed, but one can assume that once a period of negotiation has passed, the gateways are equipped with a consistent and correct model of the connectivity of the internet. At present this period of negotiation may actually take several minutes, and many TCP implementations time out within that period, but it is a design goal of the eventual algorithm that the gateway should be able to reconstruct the topology quickly enough that a TCP connection should be able to survive a failure of the route.

3. HOST ALGORITHM FOR FAULT RECOVERY

Since the gateways always attempt to have a consistent and correct model of the internetwork topology, the host strategy for fault recovery is very simple. Whenever the host feels that something is wrong, it asks the gateway for advice, and, assuming the advice is forthcoming, it believes the advice completely. The advice will be wrong only during the transient period of negotiation, which immediately follows an outage, but will otherwise be reliably correct.

In fact, it is never necessary for a host to explicitly ask a gateway for advice, because the gateway will provide it as appropriate. When a host sends a datagram to some distant net, the host should be prepared to receive back either of two advisory messages which the gateway may send. The ICMP "redirect" message indicates that the gateway to which the host sent the datagram is not longer the best gateway to reach the net in question. The gateway will have forwarded the datagram, but the host should revise its routing table to have a different immediate address for this net. The ICMP "destination unreachable" message indicates that as a result of an outage, it is currently impossible to reach the addressed net or host in any manner. On receipt of this message, a host can either abandon the connection immediately without any further retransmission, or resend slowly to see if the fault is corrected in reasonable time.

If a host could assume that these two ICMP messages would always arrive when something was amiss in the network, then no other action on the part of the host would be required in order maintain its tables in an optimal condition. Unfortunately, there are two circumstances under which the messages will not arrive properly. First, during the transient following a failure, error messages may arrive that do not correctly represent the state of the world. Thus, hosts must take an isolated error message with some scepticism. (This transient period is discussed more fully below.) Second, if the host has been sending datagrams to a particular gateway, and that gateway itself crashes, then all the other gateways in the internet will reconstruct the topology, but the gateway in question will still be down, and therefore cannot provide any advice back to the host. As long as the host continues to direct datagrams at this dead gateway, the datagrams will simply vanish off the face of the earth, and nothing will come back in return. Hosts must detect this failure.

If some gateway many hops away fails, this is not of concern to the host, for then the discovery of the failure is the responsibility of the immediate neighbor gateways, which will perform this action in a manner invisible to the host. The problem only arises if the very first gateway, the one to which the host is immediately sending the datagrams, fails. We thus identify one single task which the host must perform as its part of fault isolation in the internet: the host must use some strategy to detect that a gateway to which it is sending datagrams is dead.

Let us assume for the moment that the host implements some algorithm to detect failed gateways; we will return later to discuss what this algorithm might be. First, let us consider what the host should do when it has determined that a gateway is down. In fact, with the exception of one small problem, the action the host should take is extremely simple. The host should select some other gateway, and try sending the datagram to it. Assuming that gateway is up, this will either produce correct results, or some ICMP advice. Since we assume that, ignoring temporary periods immediately following an outage, any gateway is capable of giving correct advice, once the host has received advice from any gateway, that host is in as good a condition as it can hope to be.

There is always the unpleasant possibility that when the host tries a different gateway, that gateway too will be down. Therefore, whatever algorithm the host uses to detect a dead gateway must continuously be applied, as the host tries every gateway in turn that it knows about.

The only difficult part of this algorithm is to specify the means by which the host maintains the table of all of the gateways to which it has immediate access. Currently, the specification of the internet protocol does not architect any message by which a host can ask to be supplied with such a table. The reason is that different networks may provide very different mechanisms by which this table can be filled in.

For example, if the net is a broadcast net, such as an ethernet or a ringnet, every gateway may simply broadcast such a table from time to time, and the host need do nothing but listen to obtain the required information. Alternatively, the network may provide the mechanism of logical addressing, by which a whole set of machines can be provided with a single group address, to which a request can be sent for assistance. Failing those two schemes, the host can build up its table of neighbor gateways by remembering all the gateways from which it has ever received a message. Finally, in certain cases, it may be necessary for this table, or at least the initial entries in the table, to be constructed manually by a manager or operator at the site. In cases where the network in question provides absolutely no support for this kind of host query, at least some manual intervention will be required to get started, so that the host can find out about at least one gateway.

4. HOST ALGORITHMS FOR FAULT ISOLATION

We now return to the question raised above. What strategy should the host use to detect that it is talking to a dead gateway, so that it can know to switch to some other gateway in the list. In fact, there are several algorithms which can be used. All are reasonably simple to implement, but they have very different implications for the overhead on the host, the gateway, and the network. Thus, to a certain extent, the algorithm picked must depend on the details of the network and of the host.

NETWORK LEVEL DETECTION

Many networks, particularly the Arpanet, perform precisely the required function internal to the network. If a host sends a datagram to a dead gateway on the Arpanet, the network will return a "host dead" message, which is precisely the information the host needs to know in order to switch to another gateway. Some early implementations of Internet on the Arpanet threw these messages away. That is an exceedingly poor idea.

CONTINUOUS POLLING

The ICMP protocol provides an echo mechanism by which a host may solicit a response from a gateway. A host could simply send this message at a reasonable rate, to assure itself continuously that the gateway was still up. This works, but, since the message must be sent fairly often to detect a fault in a reasonable time, it can imply an unbearable overhead on the host itself, the network, and the gateway. This strategy is prohibited except where a specific analysis has indicated that the overhead is tolerable.

TRIGGERED POLLING

If the use of polling could be restricted to only those times when something seemed to be wrong, then the overhead would be bearable. Provided that one can get the proper advice from one's higher level protocols, it is possible to implement such a strategy. For example, one could program the TCP level so that whenever it retransmitted a segment more than once, it sent a hint down to the IP layer which triggered polling. This strategy does not have excessive overhead, but does have the problem that the host may be somewhat slow to respond to an error, since only after polling has started will the host be able to confirm that something has gone wrong, and by then the TCP above may have already timed out.

Both forms of polling suffer from a minor flaw. Hosts as well as gateways respond to ICMP echo messages. Thus, polling cannot be used to detect the error that a foreign address thought to be a gateway is actually a host. Such a confusion can arise if the physical addresses of machines are rearranged.

TRIGGERED RESELECTION

There is a strategy which makes use of a hint from a higher level, as did the previous strategy, but which avoids polling altogether. Whenever a higher level complains that the service seems to be defective, the Internet layer can pick the next gateway from the list of available gateways, and switch to it. Assuming that this gateway is up, no real harm can come of this decision, even if it was wrong, for the worst that will happen is a redirect message which instructs the host to return to the gateway originally being used. If, on the other hand, the original gateway was indeed down, then this immediately provides a new route, so the period of time until recovery is shortened. This last strategy seems particularly clever, and is probably the most generally suitable for those cases where the network itself does not provide fault isolation. (Regretably, I have forgotten who suggested this idea to me. It is not my invention.)

5. HIGHER LEVEL FAULT DETECTION

The previous discussion has concentrated on fault detection and recovery at the IP layer. This section considers what the higher layers such as TCP should do.

TCP has a single fault recovery action; it repeatedly retransmits a segment until either it gets an acknowledgement or its connection timer expires. As discussed above, it may use retransmission as an event to trigger a request for fault recovery to the IP layer. In the other direction, information may flow up from IP, reporting such things as ICMP Destination Unreachable or error messages from the attached network. The only subtle question about TCP and faults is what TCP should do when such an error message arrives or its connection timer expires.

The TCP specification discusses the timer. In the description of the open call, the timeout is described as an optional value that the client of TCP may specify; if any segment remains unacknowledged for this period, TCP should abort the connection. The default for the timeout is 30 seconds. Early TCPs were often implemented with a fixed timeout interval, but this did not work well in practice, as the following discussion may suggest.

Clients of TCP can be divided into two classes: those running on immediate behalf of a human, such as Telnet, and those supporting a program, such as a mail sender. Humans require a sophisticated response to errors. Depending on exactly what went wrong, they may want to abandon the connection at once, or wait for a long time to see if things get better. Programs do not have this human impatience, but also lack the power to make complex decisions based on details of the exact error condition. For them, a simple timeout is reasonable.

Based on these considerations, at least two modes of operation are needed in TCP. One, for programs, abandons the connection without exception if the TCP timer expires. The other mode, suitable for people, never abandons the connection on its own initiative, but reports to the layer above when the timer expires. Thus, the human user can see error messages coming from all the relevant layers, TCP and ICMP, and can request TCP to abort as appropriate. This second mode requires that TCP be able to send an asynchronous message up to its client to report the timeout, and it requires that error messages arriving at lower layers similarly flow up through TCP.

At levels above TCP, fault detection is also required. Either of the following can happen. First, the foreign client of TCP can fail, even though TCP is still running, so data is still acknowledged and the timer never expires. Alternatively, the communication path can fail, without the TCP timer going off, because the local client has no data to send. Both of these have caused trouble.

Sending mail provides an example of the first case. When sending mail using SMTP, there is an SMTP level acknowledgement that is returned when a piece of mail is successfully delivered. Several early mail receiving programs would crash just at the point where they had received all of the mail text (so TCP did not detect a timeout due to outstanding unacknowledged data) but before the mail was acknowledged at the SMTP level. This failure would cause early mail senders to wait forever for the SMTP level acknowledgement. The obvious cure was to set a timer at the SMTP level, but the first attempt to do this did not work, for there was no simple way to select the timer interval. If the interval selected was short, it expired in normal operational when sending a large file to a slow host. An interval of many minutes was needed to prevent false timeouts, but that meant that failures were detected only very slowly. The current solution in several mailers is to pick a timeout interval proportional to the size of the message.

Server telnet provides an example of the other kind of failure. It can easily happen that the communications link can fail while there is no traffic flowing, perhaps because the user is thinking. Eventually, the user will attempt to type somethinp, at which time he will discover that the connection is dead and abort it. But the host end of the connection, having nothing to send, will not discover anything wrong, and will remain waiting forever. In some systems there is no way for a user in a different process to destroy or take over such a hanging process, so there is no way to recover.

One solution to this would be to have the host server telnet query the user end now and then, to see if it is still up. (Telnet does not have an explicit query feature, but the host could negotiate some unimportant option, which should produce either agreement or disagreement in return.) The only problem with this is that a reasonable sample interval, if applied to every user on a large system, can generate an unacceptable amount of traffic and system overhead. A smart server telnet would use this query only when something seems wrong, perhaps when there had been no user activity for some time.

In both these cases, the general conclusion is that client level error detection is needed, and that the details of the mechanism are very dependent on the application. Application programmers must be made aware of the problem of failures, and must understand that error detection at the TCP or lower level cannot solve the whole problem for them.

6. KNOWING WHEN TO GIVE UP

It is not obvious, when error messages such as ICMP Destination Unreachable arrive, whether TCP should abandon the connection. The reason that error messages are difficult to interpret is that, as discussed above, after a failure of a gateway or network, there is a transient period during which the gateways may have incorrect information, so that irrelevant or incorrect error messages may sometimes return. An isolated ICMP Destination Unreachable may arrive at a host, for example, if a packet is sent during the period when the gateways are trying to find a new route. To abandon a TCP connection based on such a message arriving would be to ignore the valuable feature of the Internet that for many internal failures it reconstructs its function without any disruption of the end points.

But if failure messages do not imply a failure, what are they for? In fact, error messages serve several important purposes. First, if they arrive in response to opening a new connection, they probably are caused by opening the connection improperly (e.g., to a non-existent address) rather than by a transient network failure. Second, they provide valuable information, after the TCP timeout has occurred, as to the probable cause of the failure. Finally, certain messages, such as ICMP Parameter Problem, imply a possible implementation problem.

In general, error messages give valuable information about what went wrong, but are not to be taken as absolutely reliable. A general alertinp mechanism, such as the TCP timeout discussed above, provides a good indication that whatever is wrong is a serious condition, but without the advisory messages to augment the timer, there is no way for the client to know how to respond to the error. The combination of the timer and the advice from the error messages provide a reasonable set of facts for the client layer to have. It is important that error messages from all layers be passed up to the client module in a useful and consistent way.

MODULARITY AND EFFICIENCY IN PROTOCOL IMPLEMENTATION

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. INTRODUCTION

Many protocol implementers have made the unpleasant discovery that their packages do not run quite as fast as they had hoped. The blame for this widely observed problem has been attributed to a variety of causes, ranging from details in the design of the protocol to the underlying structure of the host operating system. This RFC will discuss some of the commonly encountered reasons why protocol implementations seem to run slowly.

Experience suggests that one of the most important factors in determining the performance of an implementation is the manner in which that implementation is modularized and integrated into the host operating system. For this reason, it is useful to discuss the question of how an implementation is structured at the same time that we consider how it will perform. In fact, this RFC will argue that modularity is one of the chief villains in attempting to obtain good performance, so that the designer is faced with a delicate and inevitable tradeoff between good structure and good performance. Further, the single factor which most strongly determines how well this conflict can be resolved is not the protocol but the operating system.

2. EFFICIENCY CONSIDERATIONS

There are many aspects to efficiency. One aspect is sending data at minimum transmission cost, which is a critical aspect of common carrier communications, if not in local area network communications. Another aspect is sending data at a high rate, which may not be possible at all if the net is very slow, but which may be the one central design constraint when taking advantage of a local net with high raw bandwidth. The final consideration is doing the above with minimum expenditure of computer resources. This last may be necessary to achieve high speed, but in the case of the slow net may be important only in that the resources used up, for example cpu cycles, are costly or otherwise needed. It is worth pointing out that these different goals often conflict; for example it is often possible to trade off efficient use of the computer against efficient use of the network. Thus, there may be no such thing as a successful general purpose protocol implementation.

The simplest measure of performance is throughput, measured in bits per second. It is worth doing a few simple computations in order to get a feeling for the magnitude of the problems involved. Assume that data is being sent from one machine to another in packets of 576 bytes, the maximum generally acceptable internet packet size. Allowing for header overhead, this packet size permits 4288 bits in each packet. If a useful throughput of 10,000 bits per second is desired, then a data bearing packet must leave the sending host about every 430 milliseconds, a little over two per second. This is clearly not difficult to achieve. However, if one wishes to achieve 100 kilobits per second throughput, the packet must leave the host every 43 milliseconds, and to achieve one megabit per second, which is not at all unreasonable on a high-speed local net, the packets must be spaced no more than 4.3 milliseconds.

These latter numbers are a slightly more alarming goal for which to set one's sights. Many operating systems take a substantial fraction of a millisecond just to service an interrupt. If the protocol has been structured as a process, it is necessary to go through a process scheduling before the protocol code can even begin to run. If any piece of a protocol package or its data must be fetched from disk, real time delays of between 30 to 100 milliseconds can be expected. If the protocol must compete for cpu resources with other processes of the system, it may be necessary to wait a scheduling quantum before the protocol can run. Many systems have a scheduling quantum of 100 milliseconds or more. Considering these sorts of numbers, it becomes immediately clear that the protocol must be fitted into the operating system in a thorough and effective manner if any like reasonable throughput is to be achieved.

There is one obvious conclusion immediately suggested by even this simple analysis. Except in very special circumstances, when many packets are being processed at once, the cost of processing a packet is dominated by factors, such as cpu scheduling, which are independent of the packet size. This suggests two general rules which any implementation ought to obey. First, send data in large packets. Obviously, if *processing time per packet* is a constant, then throughput will be directly proportional to the packet size. Second, never send an unneeded packet. Unneeded packets use up just as many resources as a packet full of data, but perform no useful function. RFC 813, "Window and Acknowledgement Strategy in TCP", discusses one aspect of reducing the number of packets sent per useful data byte. This document will mention other attacks on the same problem.

The above analysis suggests that there are two main parts to the problem of achieving good protocol performance. The first has to do with how the protocol implementation is integrated into the host operating system. The second has to do with how the protocol package itself is organized internally. This document will consider each of these topics in turn.

3. THE PROTOCOL VS. THE OPERATING SYSTEM

There are normally three reasonable ways in which to add a protocol to an operating system. The protocol can be in a process that is provided by the operating system, or it can be part of the kernel of the operating system itself, or it can be put in a separate communications processor or front end machine. This decision is strongly influenced by details of hardware architecture and operating system design; each of these three approaches has its own advantages and disadvantages.

The "process" is the abstraction which most operating systems use to provide the execution environment for user programs. A very simple path for implementing a protocol is to obtain a process from the operating system and implement the protocol to run in it. Superficially, this approach has a number of advantages. Since modifications to the kernel are not required, the job can be done by someone who is not an expert in the kernel structure. Since it is often impossible to find somebody who is experienced both in the structure of the operating system and the structure of the protocol, this path, from a management point of view, is often extremely appealing. Unfortunately, putting a protocol in a process has a number of disadvantages, related to both structure and performance. First, as was discussed above, process scheduling can be a significant source of real-time delay. There is not only the actual cost of going through the scheduler, but the problem that the operating system may not have the right sort of priority tools to bring the process into execution quickly whenever there is work to be done.

Structurally, the difficulty with putting a protocol in a process is that the protocol may be providing services, for example support of data streams, which are normally obtained by going to special kernel entry points. Depending on the generality of the operating system, it may be impossible to take a program which is accustomed to reading through a kernel entry point, and redirect it so it is reading the data from a process. The most extreme example of this problem occurs when implementing server telnet. In almost all systems, the device handler for the locally attached teletypes is located inside the kernel, and programs read and write from their teletype by making kernel calls. If server telnet is implemented in a process, it is then necessary to take the data streams provided by server telnet and somehow get them back down inside the kernel so that they mimic the interface provided by local teletypes. It is usually the case that special kernel modification is necessary to achieve this structure, which somewhat defeats the benefit of having removed the protocol from the kernel in the first place.

Clearly, then, there are advantages to putting the protocol package in the kernel. Structurally, it is reasonable to view the network as a device, and device drivers are traditionally contained in the kernel. Presumably, the problems associated with process scheduling can be sidesteped, at least to a certain extent, by placing the code inside the kernel. And it is obviously easier to make the server telnet channels mimic the local teletype channels if they are both realized in the same level in the kernel.

However, implementation of protocols in the kernel has its own set of pitfalls. First, network protocols have a characteristic which is shared by almost no other device: they require rather complex actions to be performed as a result of a timeout. The problem with this requirement is that the kernel often has no facility by which a program can be brought into execution as a result of the timer event. What is really needed, of course, is a special sort of process inside the kernel. Most systems lack this mechanism. Failing that, the only execution mechanism available is to run at interrupt time.

There are substantial drawbacks to implementing a protocol to run at interrupt time. First, the actions performed may be somewhat complex and time consuming, compared to the maximum amount of time that the operating system is prepared to spend servicing an interrupt. Problems can arise if interrupts are masked for too long. This is particularly bad when running as a result of a clock interrupt, which can imply that the clock interrupt is masked. Second, the environment provided by an interrupt handler is usually extremely primitive compared to the environment of a process.

There are usually a variety of system facilities which are unavailable while running in an interrupt handler. The most important of these is the ability to suspend execution pending the arrival of some event or message. It is a cardinal rule of almost every known operating system that one must not invoke the scheduler while running in an interrupt handler. Thus, the programmer who is forced to implement all or part of his protocol package as an interrupt handler must be the best sort of expert in the operating system involved, and must be prepared for development sessions filled with obscure bugs which crash not just the protocol package but the entire operating system.

A final problem with processing at interrupt time is that the system scheduler has no control over the percentage of system time used by the protocol handler. If a large number of packets arrive, from a foreign host that is either malfunctioning or fast, all of the time may be spent in the interrupt handler, effectively killing the system.

There are other problems associated with putting protocols into an operating system kernel. The simplest problem often encountered is that the kernel address space is simply too small to hold the piece of code in question. This is a rather artificial sort of problem, but it is a severe problem none the less in many machines. It is an appallingly unpleasant experience to do an implementation with the knowledge that for every byte of new feature put in one must find some other byte of old feature to throw out. It is hopeless to expect an effective and general implementation under this kind of constraint. Another problem is that the protocol package, once it is thoroughly entwined in the operating system, may need to be redone every time the operating system changes. If the protocol and the operating system are not maintained by the same group, this makes maintenance of the protocol package a perpetual headache.

The third option for protocol implementation is to take the protocol package and move it outside the machine entirely, on to a separate processor dedicated to this kind of task. Such a machine is often described as a communications processor or a front-end processor.

There are several advantages to this approach. First, the operating system on the communications processor can be tailored for precisely this kind of task. This makes the job of implementation much easier. Second, one does not need to redo the task for every machine to which the protocol is to be added. It may be possible to reuse the same front-end machine on different host computers. Since the task need not be done as many times, one might hope that more attention could be paid to doing it right. Given a careful implementation in an environment which is optimized for this kind of task, the resulting package should turn out to be very efficient.

Unfortunately, there are also problems with this approach. There is, of course, a financial problem associated with buying an additional computer. In many cases, this is not a problem at all since the cost is negligible compared to what the programmer would cost to do the job in the mainframe itself. More fundamentally, the communications processor approach does not completely sidestep any of the problems raised above. The reason is that the communications processor, since it is a separate machine, must be attached to the mainframe by some mechanism. Whatever that mechanism, code is required in the mainframe to deal with it. It can be argued that the program to deal with the communications processor is simpler than the program to implement the entire protocol package. Even if that is so, the communications processor interface package is still a protocol in nature, with all of the same structural problems. Thus, all of the issues raised above must still be faced. In addition to those problems, there are some other, more subtle problems associated with an outboard implementation of a protocol. We will return to these problems later.

There is a way of attaching a communications processor to a mainframe host which sidesteps all of the mainframe implementation problems, which is to use some preexisting interface on the host machine as the port by which a communications processor is attached. This strategy is often used as a last stage of desperation when the software on the host computer is so intractable that it cannot be changed in any way. Unfortunately, it is almost inevitably the case that all of the available interfaces are totally unsuitable for this purpose, so the result is unsatisfactory at best.

The most common way in which this form of attachment occurs is when a network connection is being used to mimic local teletypes. In this case, the front-end processor can be attached to the mainframe by simply providing a number of wires out of the front-end processor, each corresponding to a connection, which are plugged into teletype ports on the mainframe computer. (Because of the appearance of the physical configuration which results from this arrangement, Michael Padlipsky has described this as the "milking machine" approach to computer networking.) This strategy solves the immediate problem of providing remote access to a host, but it is extremely inflexible. The channels being provided to the host are restricted by the host software to one purpose only, remote login. It is impossible to use them for any other purpose, such as file transfer or sending mail, so the host is integrated into the network environment in an extremely limited and inflexible manner. If this is the best that can be done, then it should be tolerated. Otherwise, implementors should be strongly encouraged to take a more flexible approach.

4. PROTOCOL LAYERING

The previous discussion suggested that there was a decision to be made as to where a protocol ought to be implemented. In fact, the decision is much more complicated than that, for the goal is not to implement a single protocol, but to implement a whole family of protocol layers, starting with a device driver or local network driver at the bottom, then IP and TCP, and eventually reaching the application specific protocol, such as Telnet, FTP and SMTP on the top. Clearly, the bottommost of these layers is somewhere within the kernel, since the physical device driver for the net is almost inevitably located there. Equally clearly, the top layers of this package, which provide the user his ability to perform the remote login function or to send mail, are not entirely contained within the kernel. Thus, the question is not whether the protocol family shall be inside or outside the kernel, but how it shall be sliced in two between that part inside and that part outside.

Since protocols come nicely layered, an obvious proposal is that one of the layer interfaces should be the point at which the inside and outside components are sliced apart. Most systems have been implemented in this way, and many have been made to work quite effectively. One obvious place to slice is at the upper interface of TCP. Since TCP provides a bidirectional byte stream, which is somewhat similar to the I/O facility provided by most operating systems, it is possible to make the interface to TCP almost mimic the interface to other existing devices. Except in the matter of opening a connection, and dealing with peculiar failures, the software using TCP need not know that it is a network connection, rather than a local I/O stream that is providing the communications function. This approach does put TCP inside the kernel, which raises all the problems addressed above. It also raises the problem that the interface to the IP layer can, if the programmer is not careful, become excessively buried inside the kernel. It must be remembered that things other than TCP are expected to run on top of IP. The IP interface must be made accessible, even if TCP sits on top of it inside the kernel.

Another obvious place to slice is above Telnet. The advantage of slicing above Telnet is that it solves the problem of having remote login channels emulate local teletype channels. The disadvantage of putting Telnet into the kernel is that the amount of code which has now been included there is getting remarkably large. In some early implementations, the size of the network package, when one includes protocols at the level of Telnet, rivals the size of the rest of the supervisor. This leads to vague feelings that all is not right.

Any attempt to slice through a lower layer boundary, for example between internet and TCP, reveals one fundamental problem. The TCP layer, as well as the IP layer, performs a demultiplexing function on incoming datagrams. Until the TCP header has been examined, it is not possible to know for which user the packet is ultimately destined. Therefore, if TCP, as a whole, is moved outside the kernel, it is necessary to create one separate process called the TCP process, which performs the TCP multiplexing function, and probably all of the rest of TCP processing as well. This means that incoming data destined for a user process involves not just a scheduling of the user process, but scheduling the TCP process first.

This suggests an alternative structuring strategy which slices through the protocols, not along an established layer boundary, but along a functional boundary having to do with demultiplexing. In this approach, certain parts of IP and certain parts of TCP are placed in the kernel. The amount of code placed there is sufficient so that when an incoming datagram arrives, it is possible to know for which process that datagram is ultimately destined. The datagram is then routed directly to the final process, where additional IP and TCP processing is performed on it. This removes from the kernel any requirement for timer based actions, since they can be done by the process provided by the user. This structure has the additional advantage of reducing the amount of code required in the kernel, so that it is suitable for systems where kernel space is at a premium. The RFC 814, titled "Names, Addresses, Ports, and Routes," discusses this rather orthogonal slicing strategy in more detail.

A related discussion of protocol layering and multiplexing can be found in Cohen and Postel [1].

5. BREAKING DOWN THE BARRIERS

In fact, the implementor should be sensitive to the possibility of even more peculiar slicing strategies in dividing up the various protocol layers between the kernel and the one or more user processes. The result of the strategy proposed above was that part of TCP should execute in the process of the user. In other words, instead of having one TCP process for the system, there is one TCP process per connection. Given this architecture, it is not longer necessary to imagine that all of the TCPs are identical. One TCP could be optimized for high throughput applications, such as file transfer. Another TCP could be optimized for small low delay applications such as Telnet. In fact, it would be possible to produce a TCP which was somewhat integrated with the Telnet or FTP on top of it. Such an integration is extremely important, for it can lead to a kind of efficiency which more traditional structures are incapable of producing.

Earlier, this paper pointed out that one of the important rules to achieving efficiency was to send the minimum number of packets for a given amount of data. The idea of protocol layering interacts very strongly (and poorly) with this goal, because independent layers have independent ideas about when packets should be sent, and unless these layers can somehow be brought into cooperation, additional packets will flow. The best example of this is the operation of server telnet in a character at a time remote echo mode on top of TCP. When a packet containing a character arrives at a server host, each layer has a different response to that packet. TCP has an obligation to acknowledge the packet. Either server telnet or the application layer above has an obligation to echo the character received in the packet. If the character is a Telnet control sequence, then Telnet has additional actions which it must perform in response to the packet. The result of this, in most implementations, is that several packets are sent back in response to the one arriving packet.

Combining all of these return messages into one packet is important for several reasons. First, of course, it reduces the number of packets being sent over the net, which directly reduces the charges incurred for many common carrier tariff structures. Second, it reduces the number of scheduling actions which will occur inside both hosts, which, as was discussed above, is extremely important in improving throughput.

The way to achieve this goal of packet sharing is to break down the barrier between the layers of the protocols, in a very restrained and careful manner, so that a limited amount of information can leak across the barrier to enable one layer to optimize its behavior with respect to the desires of the layers above and below it. For example, it would represent an improvement if TCP, when it received a packet, could ask the layer above whether or not it would be worth pausing for a few milliseconds before sending an acknowledgement in order to see if the upper layer would have any outgoing data to send. Dallying before sending the acknowledgement produces precisely the right sort of optimization if the client of TCP is server Telnet. However, dallying before sending an acknowledgement is absolutely unacceptable if TCP is being used for file transfer, for in file transfer there is almost never data flowing in the reverse direction, and the delay in sending the acknowledgement probably translates directly into a delay in obtaining the next packets. Thus, TCP must know a little about the layers above it to adjust its performance as needed.

It would be possible to imagine a general purpose TCP which was equipped with all sorts of special mechanisms by which it would query the layer above and modify its behavior accordingly. In the structures suggested above, in which there is not one but several TCPs, the TCP can simply be modified so that it produces the correct behavior as a matter of course. This structure has the disadvantage that there will be several implementations of TCP existing on a single machine, which can mean more maintenance headaches if a problem is found where TCP needs to be changed. However, it is probably the case that each of the TCPs will be substantially simpler than the general purpose TCP which would otherwise have been built. There are some experimental projects currently under way which suggest that this approach may make designing of a TCP, or almost any other layer, substantially easier, so that the total effort involved in bringing up a complete package is actually less if this approach is followed. This approach is by no means generally accepted, but deserves some consideration.

The general conclusion to be drawn from this sort of consideration is that a layer boundary has both a benefit and a penalty. A visible layer boundary, with a well specified interface, provides a form of isolation between two layers which allows one to be changed with the confidence that the other one will not stop working as a result. However, a firm layer boundary almost inevitably leads to inefficient operation. This can easily be seen by analogy with other aspects of operating systems.

Consider, for example, file systems. A typical operating system provides a file system, which is a highly abstracted representation of a disk. The interface is highly formalized, and presumed to be highly stable. This makes it very easy for naive users to have access to disks without having to write a great deal of software. The existence of a file system is clearly beneficial. On the other hand, it is clear that the restricted interface to a file system almost inevitably leads to inefficiency. If the interface is organized as a sequential read and write of bytes, then there will be people who wish to do high throughput transfers who cannot achieve their goal. If the interface is a virtual memory interface, then other users will regret the necessity of building a byte stream interface on top of the memory mapped file. The most objectionable inefficiency results when a highly sophisticated package, such as a data base management package, must be built on top of an existing operating system. Almost inevitably, the implementors of the database system attempt to reject the file system and obtain direct access to the disks. They have sacrificed modularity for efficiency.

The same conflict appears in networking, in a rather extreme form. The concept of a protocol is still unknown and frightening to most naive programmers. The idea that they might have to implement a protocol, or even part of a protocol, as part of some application package, is a dreadful thought. And thus there is great pressure to hide the function of the net behind a very hard barrier. On the other hand, the kind of inefficiency which results from this is a particularly undesirable sort of inefficiency, for it shows up, among other things, in increasing the cost of the communications resource used up to achieve the application goal. In cases where one must pay for one's communications costs, they usually turn out to be the dominant cost within the system. Thus, doing an excessively good job of packaging up the protocols in an inflexible manner has a direct impact on increasing the cost of the critical resource within the system.

This is a dilemma which will probably only be solved when programmers become somewhat less alarmed about protocols, so that they are willing to weave a certain amount of protocol structure into their application program, much as application programs today weave parts of database management systems into the structure of their application program.

An extreme example of putting the protocol package behind a firm layer boundary occurs when the protocol package is relegated to a front- end processor. In this case the interface to the protocol is some other protocol. It is difficult to imagine how to build close cooperation between layers when they are that far separated. Realistically, one of the prices which must be associated with an implementation so physically modularized is that the performance will suffer as a result. Of course, a separate processor for protocols could be very closely integrated into the mainframe architecture, with interprocessor co-ordination signals, shared memory, and similar features. Such a physical modularity might work very well, but there is little documented experience with this closely coupled architecture for protocol support.

6. EFFICIENCY OF PROTOCOL PROCESSING

To this point, this document has considered how a protocol package should be broken into modules, and how those modules should be distributed between free standing machines, the operating system kernel, and one or more user processes. It is now time to consider the other half of the efficiency question, which is what can be done to speed the execution of those programs that actually implement the protocols. We will make some specific observations about TCP and IP, and then conclude with a few generalities.

IP is a simple protocol, especially with respect to the processing of normal packets, so it should be easy to get it to perform efficiently. The only area of any complexity related to actual packet processing has to do with fragmentation and reassembly. The reader is referred to RFC 815, titled "IP Datagram Reassembly Algorithms", for specific consideration of this point.

Most costs in the IP layer come from table look up functions, as opposed to packet processing functions. An outgoing packet requires two translation functions to be performed. The internet address must be translated to a target gateway, and a gateway address must be translated to a local network number (if the host is attached to more than one network). It is easy to build a simple implementation of these table look up functions that in fact performs very poorly. The programmer should keep in mind that there may be as many as a thousand network numbers in a typical configuration. Linear searching of a thousand entry table on every packet is extremely unsuitable. In fact, it may be worth asking TCP to cache a hint for each connection, which can be handed down to IP each time a packet is sent, to try to avoid the overhead of a table look up.

TCP is a more complex protocol, and presents many more opportunities for getting things wrong. There is one area which is generally accepted as causing noticeable and substantial overhead as part of TCP processing. This is computation of the checksum. It would be nice if this cost could be avoided somehow, but the idea of an end- to-end checksum is absolutely central to the functioning of TCP. No host implementor should think of omitting the validation of a checksum on incoming data.

Various clever tricks have been used to try to minimize the cost of computing the checksum. If it is possible to add additional microcoded instructions to the machine, a checksum instruction is the most obvious candidate. Since computing the checksum involves picking up every byte of the segment and examining it, it is possible to combine the operation of computing the checksum with the operation of copying the segment from one location to another. Since a number of data copies are probably already required as part of the processing structure, this kind of sharing might conceivably pay off if it didn't cause too much trouble to the modularity of the program. Finally, computation of the checksum seems to be one place where careful attention to the details of the algorithm used can make a drastic difference in the throughput of the program.

The Multics system provides one of the best case studies of this, since Multics is about as poorly organized to perform this function as any machine implementing TCP. Multics is a 36-bit word machine, with four 9-bit bytes per word. The eight-bit bytes of a TCP segment are laid down packed in memory, ignoring word boundaries. This means that when it is necessary to pick up the data as a set of 16-bit units for the purpose of adding them to compute checksums, horrible masking and shifting is required for each 16-bit value. An early version of a program using this strategy required 6 milliseconds to checksum a 576-byte segment. Obviously, at this point, checksum computation was becoming the central bottleneck to throughput. A more careful recoding of this algorithm reduced the checksum processing time to less than one millisecond. The strategy used was extremely dirty. It involved adding up carefully selected words of the area in which the data lay, knowing that for those particular words, the 16-bit values were properly aligned inside the words. Only after the addition had been done were the various sums shifted, and finally added to produce the eventual checksum.

This kind of highly specialized programming is probably not acceptable if used everywhere within an operating system. It is clearly appropriate for one highly localized function which can be clearly identified as an extreme performance bottleneck.

Another area of TCP processing which may cause performance problems is the overhead of examining all of the possible flags and options which occur in each incoming packet. One paper, by Bunch and Day [2], asserts that the overhead of packet header processing is actually an important limiting factor in throughput computation. Not all measurement experiments have tended to support this result. To whatever extent it is true, however, there is an obvious strategy which the implementor ought to use in designing his program. He should build his program to optimize the expected case.

It is easy, especially when first designing a program, to pay equal attention to all of the possible outcomes of every test. In practice, however, few of these will ever happen. A TCP should be built on the assumption that the next packet to arrive will have absolutely nothing special about it, and will be the next one expected in the sequence space. One or two tests are sufficient to determine that the expected set of control flags are on. (The ACK flag should be on; the Push flag may or may not be on. No other flags should be on.) One test is sufficient to determine that the sequence number of the incoming packet is one greater than the last sequence number received. In almost every case, that will be the actual result.

Again, using the Multics system as an example, failure to optimize the case of receiving the expected sequence number had a detectable effect on the performance of the system. The particular problem arose when a number of packets arrived at once. TCP attempted to process all of these packets before awaking the user. As a result, by the time the last packet arrived, there was a threaded list of packets which had several items on it. When a new packet arrived, the list was searched to find the location into which the packet should be inserted. Obviously, the list should be searched from highest sequence number to lowest sequence number, because one is expecting to receive a packet which comes after those already received. By mistake, the list was searched from front to back, starting with the packets with the lowest sequence number. The amount of time spent searching this list backwards was easily detectable in the metering measurements.

Other data structures can be organized to optimize the action which is normally taken on them. For example, the retransmission queue is very seldom actually used for retransmission, so it should not be organized to optimize that action. In fact, it should be organized to optimized the discarding of things from it when the acknowledgement arrives. In many cases, the easiest way to do this is not to save the packet at all, but to reconstruct it only if it needs to be retransmitted, starting from the data as it was originally buffered by the user.

There is another generality, at least as important as optimizing the common case, which is to avoid copying data any more times than necessary. One more result from the Multics TCP may prove enlightening here. Multics takes between two and three milliseconds within the TCP layer to process an incoming packet, depending on its size. For a 576- byte packet, the three milliseconds is used up approximately as follows. One millisecond is used computing the checksum. Six hundred microseconds is spent copying the data. (The data is copied twice, at .3 milliseconds a copy.) One of those copy operations could correctly be included as part of the checksum cost, since it is done to get the data on a known word boundary to optimize the checksum algorithm. However, the copy also performs another necessary transfer at the same time. Header processing and packet resequencing takes .7 milliseconds. The rest of the time is used in miscellaneous processing, such as removing packets from the retransmission queue which are acknowledged by this packet.

Data copying is the second most expensive single operation after data checksuming. Some implementations, often because of an excessively layered modularity, end up copying the data around a great deal. Other implementations end up copying the data because there is no shared memory between processes, and the data must be moved from process to process via a kernel operation. Unless the amount of this activity is kept strictly under control, it will quickly become the major performance bottleneck.

7. CONCLUSIONS

This document has addressed two aspects of obtaining performance from a protocol implementation, the way in which the protocol is layered and integrated into the operating system, and the way in which the detailed handling of the packet is optimized.

It would be nice if one or the other of these costs would completely dominate, so that all of one's attention could be concentrated there. Regrettably, this is not so. Depending on the particular sort of traffic one is getting, for example, whether Telnet one-byte packets or file transfer maximum size packets at maximum speed, one can expect to see one or the other cost being the major bottleneck to throughput. Most implementors who have studied their programs in an attempt to find out where the time was going have reached the unsatisfactory conclusion that it is going equally to all parts of their program. With the possible exception of checksum processing, very few people have ever found that their performance problems were due to a single, horrible bottleneck which they could fix by a single stroke of inventive programming. Rather, the performance was something which was improved by painstaking tuning of the entire program.

Most discussions of protocols begin by introducing the concept of layering, which tends to suggest that layering is a fundamentally wonderful idea which should be a part of every consideration of protocols. In fact, layering is a mixed blessing.

Clearly, a layer interface is necessary whenever more than one client of a particular layer is to be allowed to use that same layer. But an interface, precisely because it is fixed, inevitably leads to a lack of complete understanding as to what one layer wishes to obtain from another. This has to lead to inefficiency.

Furthermore, layering is a potential snare in that one is tempted to think that a layer boundary, which was an artifact of the specification procedure, is in fact the proper boundary to use in modularizing the implementation. Again, in certain cases, an architected layer must correspond to an implemented layer, precisely so that several clients can have access to that layer in a reasonably straightforward manner. In other cases, cunning rearrangement of the implemented module boundaries to match with various functions, such as the demultiplexing of incoming packets, or the sending of asynchronous outgoing packets, can lead to unexpected performance improvements compared to more traditional implementation strategies.

Finally, good performance is something which is difficult to retrofit onto an existing program. Since performance is influenced, not just by the fine detail, but by the gross structure, it is sometimes the case that in order to obtain a substantial performance improvement, it is necessary to completely redo the program from the bottom up. This is a great disappointment to programmers, especially those doing a protocol implementation for the first time. Programmers who are somewhat inexperienced and unfamiliar with protocols are sufficiently concerned with getting their program logically correct that they do not have the capacity to think at the same time about the performance of the structure they are building. Only after they have achieved a logically correct program do they discover that they have done so in a way which has precluded real performance.

Clearly, it is more difficult to design a program thinking from the start about both logical correctness and performance. With time, as implementors as a group learn more about the appropriate structures to use for building protocols, it will be possible to proceed with an implementation project having more confidence that the structure is rational, that the program will work, and that the program will work well. Those of us now implementing protocols have the privilege of being on the forefront of this learning process. It should be no surprise that our programs sometimes suffer from the uncertainty we bring to bear on them.

CITATIONS

[1] Cohen and Postel, "On Protocol Multiplexing", Sixth Data Communications Symposium, ACM/IEEE, November 1979.

[2] Bunch and Day, "Control Structure Overhead in TCP", Trends and Applications: Computer Networking, NBS Symposium, May 1980.

A Protocol for Packet Network Intercommunication

VINTON G. CERF AND ROBERT E. KAHN, MEMBER, IEEE

Abstract—**A protocol that supports the sharing of resources that exist in different packet switching networks is presented. The protocol provides for variation in individual network packet sizes, transmission failures, sequencing, flow control, end-to-end error checking, and the creation and destruction of logical process-to-process connections. Some implementation issues are considered, and problems such as internetwork routing, accounting, and timeouts are exposed.**

INTRODUCTION

IN THE LAST few years considerable effort has been expended on the design and implementation of packet switching networks [1]-[7],[14],[17]. A principle reason for developing such networks has been to facilitate the sharing of computer resources. A packet communication network includes a transportation mechanism for delivering data between computers or between computers and terminals. To make the data meaningful, computers and terminals share a common protocol (i.e., a set of agreed upon conventions). Several protocols have already been developed for this purpose [8]-[12],[16]. However, these protocols have addressed only the problem of communication on the same network. In this paper we present a protocol design and philosophy that supports the sharing of resources that exist in different packet switching networks.

After a brief introduction to internetwork protocol issues, we describe the function of a GATEWAY as an interface between networks and discuss its role in the protocol. We then consider the various details of the protocol, including addressing, formatting, buffering, sequencing, flow control, error control, and so forth. We close with a description of an interprocess communication mechanism and show how it can be supported by the internetwork protocol.

Even though many different and complex problems must be solved in the design of an individual packet switching network, these problems are manifestly compounded when dissimilar networks are interconnected. Issues arise which may have no direct counterpart in an individual network and which strongly influence the way in which internetwork communication can take place.

A typical packet switching network is composed of a set of computer resources called HOSTS, a set of one or more *packet switches*, and a collection of communication media that interconnect the packet switches. Within each HOST, we assume that there exist *processes* which must communicate with processes in their own or other HOSTS. Any current de[illegible]on of a process will be adequate for our purposes [13]. [illegible]ese processes are generally the ultimate source and destination of data in the network. Typically, within an individual network, there exists a protocol for communication between any source and destination process. Only the source and destination processes require knowledge of this convention for communication to take place. Processes in two distinct networks would ordinarily use different protocols for this purpose. The ensemble of packet switches and communication media is called the *packet switching subnet*. Fig. 1 illustrates these ideas.

In a typical packet switching subnet, data of a fixed maximum size are accepted from a source HOST, together with a formatted destination address which is used to route the data in a store and forward fashion. The transmit time for this data is usually dependent upon internal network parameters such as communication media data rates, buffering and signaling strategies, routing, propagation delays, etc. In addition, some mechanism is generally present for error handling and determination of status of the networks components.

Individual packet switching networks may differ in their implementations as follows.

1) Each network may have distinct ways of addressing the receiver, thus requiring that a uniform addressing scheme be created which can be understood by each individual network.

2) Each network may accept data of different maximum size, thus requiring networks to deal in units of the smallest maximum size (which may be impractically small) or requiring procedures which allow data crossing a network boundary to be reformatted into smaller pieces.

3) The success or failure of a transmission and its performance in each network is governed by different time delays in accepting, delivering, and transporting the data. This requires careful development of internetwork timing procedures to insure that data can be successfully delivered through the various networks.

4) Within each network, communication may be disrupted due to unrecoverable mutation of the data or missing data. End-to-end restoration procedures are desirable to allow complete recovery from these conditions.

Paper approved by the Associate Editor for Data Communications of the IEEE Communications Society for publication without oral presentation. Manuscript received November 5, 1973. The research reported in this paper was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC 15-73-C-0370.

V. G. Cerf is with the Department of Computer Science and Electrical Engineering, Stanford University, Stanford, Calif.

R. E. Kahn is with the Information Processing Technology Office, Advanced Research Projects Agency, Department of Defense, Arlington, Va.

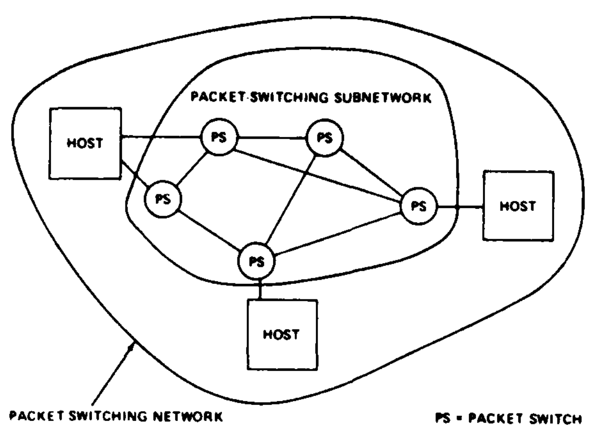


Fig. 1. Typical packet switching network.

5) Status information, routing, fault detection, and isolation are typically different in each network. Thus, to obtain verification of certain conditions, such as an inaccessible or dead destination, various kinds of coordination must be invoked between the communicating networks.

It would be extremely convenient if all the differences between networks could be economically resolved by suitable interfacing at the network boundaries. For many of the differences, this objective can be achieved. However, both economic and technical considerations lead us to prefer that the interface be as simple and reliable as possible and deal primarily with passing data between networks that use different packet switching strategies.

The question now arises as to whether the interface ought to account for differences in HOST or process level protocols by transforming the source conventions into the corresponding destination conventions. We obviously want to allow conversion between packet switching strategies at the interface, to permit interconnection of existing and planned networks. However, the complexity and dissimilarity of the HOST or process level protocols makes it desirable to avoid having to transform between them at the interface, even if this transformation were always possible. Rather, compatible HOST and process level protocols must be developed to achieve effective internetwork resource sharing. The unacceptable alternative is for every HOST or process to implement every protocol (a potentially unbounded number) that may be needed to communicate with other networks. We therefore assume that a common protocol is to be used between HOST's or processes in different networks and that the interface between networks should take as small a role as possible in this protocol.

To allow networks under different ownership to interconnect, some accounting will undoubtedly be needed for traffic that passes across the interface. In its simplest terms, this involves an accounting of packets handled by each net for which charges are passed from net to net until the buck finally stops at the user or his representative. Furthermore, the interconnection must preserve intact the internal operation of each individual network. This is easily achieved if two networks interconnect as if each were a HOST to the other network, but without utilizing or indeed incorporating any elaborate HOST protocol transformations.

It is thus apparent that the interface between networks must play a central role in the development of any network interconnection strategy. We give a special name to this interface that performs these functions and call it a GATEWAY.

THE GATEWAY NOTION

In Fig. 2 we illustrate three individual networks labeled A, B, and C which are joined by GATEWAYS M and N. GATEWAY M interfaces network A with network B, and GATEWAY N interfaces network B to network C. We assume that an individual network may have more than one GATEWAY (e.g., network B) and that there may be more than one GATEWAY path to use in going between a pair of networks. The responsibility for properly routing data resides in the GATEWAY.

In practice, a GATEWAY between two networks may be composed of two halves, each associated with its own network. It is possible to implement each half of a GATEWAY so it need only embed internetwork packets in local packet format or extract them. We propose that the GATEWAYS handle internetwork packets in a standard format, but we are not proposing any particular transmission procedure between GATEWAY halves.

Let us now trace the flow of data through the interconnected networks. We assume a packet of data from process X enters network A destined for process Y in network C. The address of Y is initially specified by process X and the address of GATEWAY M is derived from the address of process Y. We make no attempt to specify whether the choice of GATEWAY is made by process X, its HOST, or one of the packet switches in network A. The packet traverses network A until it reaches GATEWAY M. At the GATEWAY, the packet is reformatted to meet the requirements of network B, account is taken of this unit of flow between A and B, and the GATEWAY delivers the packet to network B. Again the derivation of the next GATEWAY address is accomplished based on the address of the destination Y. In this case, GATEWAY N is the next one. The packet traverses network B until it finally reaches GATEWAY N where it is formatted to meet the requirements of network C. Account is again taken of this unit of flow between networks B and C. Upon entering network C, the packet is routed to the HOST in which process Y resides and there it is delivered to its ultimate destination.

Since the GATEWAY must understand the address of the source and destination HOSTS, this information must be available in a standard format in every packet which arrives at the GATEWAY. This information is contained in an *internetwork header* prefixed to the packet by the source HOST. The packet format, including the internet-

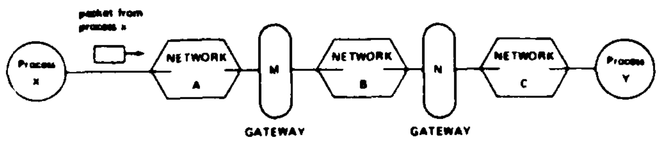


Fig. 2. Three networks interconnected by two GATEWAYS.

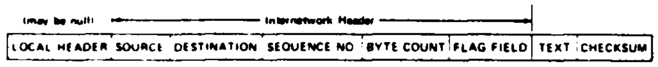


Fig. 3. Internetwork packet format (fields not shown to scale).

work header, is illustrated in Fig. 3. The source and destination entries uniformly and uniquely identify the address of every HOST in the composite network. Addressing is a subject of considerable complexity which is discussed in greater detail in the next section. The next two entries in the header provide a sequence number and a byte count that may be used to properly sequence the packets upon delivery to the destination and may also enable the GATEWAYS to detect fault conditions affecting the packet. The flag field is used to convey specific control information and is discussed in the section on retransmission and duplicate detection later. The remainder of the packet consists of text for delivery to the destination and a trailing check sum used for end-to-end software verification. The GATEWAY does *not* modify the text and merely forwards the check sum along without computing or recomputing it.

Each network may need to augment the packet format before it can pass through the individual network. We have indicated a *local header* in the figure which is prefixed to the beginning of the packet. This local header is introduced merely to illustrate the concept of embedding an internetwork packet in the format of the individual network through which the packet must pass. It will obviously vary in its exact form from network to network and may even be unnecessary in some cases. Although not explicitly indicated in the figure, it is also possible that a local trailer may be appended to the end of the packet.

Unless all transmitted packets are legislatively restricted to be small enough to be accepted by every individual network, the GATEWAY may be forced to split a packet into two or more smaller packets. This action is called fragmentation and must be done in such a way that the destination is able to piece together the fragmented packet. It is clear that the internetwork header format imposes a minimum packet size which all networks must carry (obviously all networks will want to carry packets larger than this minimum). We believe the long range growth and development of internetwork communication would be seriously inhibited by specifying how much larger than the minimum a packet size can be, for the following reasons.

1) If a maximum permitted packet size is specified then it becomes impossible to completely isolate the internal packet size parameters of one network from the internal packet size parameters of all other networks.

2) It would be very difficult to increase the maximum permitted packet size in response to new technology (e.g., large memory systems, higher data rate communication facilities, etc.) since this would require the agreement and then implementation by all participating networks.

3) Associative addressing and packet encryption may require the size of a particular packet to expand during transit for incorporation of new information.

Provision for fragmentation (regardless of where it is performed) permits packet size variations to be handled on an individual network basis without global administration and also permits HOSTS and processes to be insulated from changes in the packet sizes permitted in any networks through which their data must pass.

If fragmentation must be done, it appears best to do it upon entering the next network at the GATEWAY since only this GATEWAY (and not the other networks) must be aware of the internal packet size parameters which made the fragmentation necessary.

If a GATEWAY fragments an incoming packet into two or more packets, they must eventually be passed along to the destination HOST as fragments or reassembled for the HOST. It is conceivable that one might desire the GATEWAY to perform the reassembly to simplify the task of the destination HOST (or process) and/or to take advantage of a larger packet size. We take the position that GATEWAYS should not perform this function since GATEWAY reassembly can lead to serious buffering problems, potential deadlocks, the necessity for all fragments of a packet to pass through the same GATEWAY, and increased delay in transmission. Furthermore, it is not sufficient for the GATEWAYS to provide this function since the final GATEWAY may also have to fragment a packet for transmission. Thus the destination HOST must be prepared to do this task.

Let us now turn briefly to the somewhat unusual accounting effect which arises when a packet may be fragmented by one or more GATEWAYS. We assume, for simplicity, that each network initially charges a fixed rate per packet transmitted, regardless of distance, and if one network can handle a larger packet size than another, it charges a proportionally larger price per packet. We also assume that a subsequent increase in any network's packet size does not result in additional cost per packet to its users. The charge to a user thus remains basically constant through any net which must fragment a packet. The unusual effect occurs when a packet is fragmented into smaller packets which must individually pass through a subsequent network with a larger packet size than the original unfragmented packet. We expect that most networks will naturally select packet sizes close to one another, but in any case, an increase in packet size in one net, even when it causes fragmentation, will not increase the cost of transmission and may actually decrease it. In the event that any other packet charging policies (than the one we suggest) are adopted, differences in cost can be used as an economic lever toward optimization of individual network performance.

PROCESS LEVEL COMMUNICATION

We suppose that processes wish to communicate in full duplex with their correspondents using unbounded but finite length messages. A single character might constitute the text of a message from a process to a terminal or vice versa. An entire page of characters might constitute the text of a message from a file to a process. A data stream (e.g., a continuously generated bit string) can be represented as a sequence of finite length messages.

Within a HOST we assume the existence of a transmission control program (TCP) which handles the transmission and acceptance of messages on behalf of the processes it serves. The TCP is in turn served by one or more packet switches connected to the HOST in which the TCP resides. Processes that want to communicate present messages to the TCP for transmission, and TCP's deliver incoming messages to the appropriate destination processes. We allow the TCP to break up messages into segments because the destination may restrict the amount of data that may arrive, because the local network may limit the maximum transmission size, or because the TCP may need to share its resources among many processes concurrently. Furthermore, we constrain the length of a segment to an integral number of 8-bit bytes. This uniformity is most helpful in simplifying the software needed with HOST machines of different natural word lengths. Provision at the process level can be made for padding a message that is not an integral number of bytes and for identifying which of the arriving bytes of text contain information of interest to the receiving process.

Multiplexing and demultiplexing of segments among processes are fundamental tasks of the TCP. On transmission, a TCP must multiplex together segments from different source processes and produce internetwork packets for delivery to one of its serving packet switches. On reception, a TCP will accept a sequence of packets from its serving packet switch(es). From this sequence of arriving packets (generally from different HOSTS), the TCP must be able to reconstruct and deliver messages to the proper destination processes.

We assume that every segment is augmented with additional information that allows transmitting and receiving TCP's to identify destination and source processes, respectively. At this point, we must face a major issue. How should the source TCP format segments destined for the same destination TCP? We consider two cases.

Case 1): If we take the position that segment boundaries are immaterial and that a byte stream can be formed of segments destined for the same TCP, then we may gain improved transmission efficiency and resource sharing by arbitrarily parceling the stream into packets, permitting many segments to share a single internetwork packet header. However, this position results in the need to reconstruct exactly, and in order, the stream of text bytes produced by the source TCP. At the destination, this stream must first be parsed into segments and these in turn must be used to reconstruct messages for delivery to the appropriate processes.

There are fundamental problems associated with this strategy due to the possible arrival of packets out of order at the destination. The most critical problem appears to be the amount of interference that processes sharing the same TCP-TCP byte stream may cause among themselves. This is especially so at the receiving end. First, the TCP may be put to some trouble to parse the stream back into segments and then distribute them to buffers where messages are reassembled. If it is not readily apparent that all of a segment has arrived (remember, it may come as several packets), the receiving TCP may have to suspend parsing temporarily until more packets have arrived. Second, if a packet is missing, it may not be clear whether succeeding segments, even if they are identifiable, can be passed on to the receiving process, unless the TCP has knowledge of some process level sequencing scheme. Such knowledge would permit the TCP to decide whether a succeeding segment could be delivered to its waiting process. Finding the beginning of a segment when there are gaps in the byte stream may also be hard.

Case 2): Alternatively, we might take the position that the destination TCP should be able to determine, upon its arrival and without additional information, for which process or processes a received packet is intended, and if so, whether it should be delivered then.

If the TCP is to determine for which process an arriving packet is intended, every packet must contain a *process header* (distinct from the internetwork header) that completely identifies the destination process. For simplicity, we assume that each packet contains text from a single process which is destined for a single process. Thus each packet need contain only one process header. To decide whether the arriving data is deliverable to the destination process, the TCP must be able to determine whether the data is in the proper sequence (we can make provision for the destination process to instruct its TCP to ignore sequencing, but this is considered a special case). With the assumption that each arriving packet contains a process header, the necessary sequencing and destination process identification is immediately available to the destination TCP.

Both Cases 1) and 2) provide for the demultiplexing and delivery of segments to destination processes, but only Case 2) does so without the introduction of potential interprocess interference. Furthermore, Case 1) introduces extra machinery to handle flow control on a HOST-to-HOST basis, since there must also be some provision for process level control, and this machinery is little used since the probability is small that within a given HOST, two processes will be coincidentally scheduled to send messages to the same destination HOST. For this reason, we select the method of Case 2) as a part of the *internetwork transmission protocol.*

ADDRESS FORMATS

The selection of address formats is a problem between networks because the local network addresses of TCP's may vary substantially in format and size. A uniform internetwork TCP address space, understood by each GATEWAY and TCP, is essential to routing and delivery of internetwork packets.

Similar troubles are encountered when we deal with process addressing and, more generally, port addressing. We introduce the notion of *ports* in order to permit a process to distinguish between multiple message streams. The port is simply a designator of one such message stream associated with a process. The means for identifying a port are generally different in different operating systems, and therefore, to obtain uniform addressing, a standard port address format is also required. A port address designates a full duplex message stream.

TCP ADDRESSING

TCP addressing is intimately bound up in routing issues, since a HOST or GATEWAY must choose a suitable destination HOST or GATEWAY for an outgoing internetwork packet. Let us postulate the following address format for the TCP address (Fig. 4). The choice for network identification (8 bits) allows up to 256 distinct networks. This size seems sufficient for the forseeable future. Similarly, the TCP identifier field permits up to 65 536 distinct TCP's to be addressed, which seems more than sufficient for any given network.

As each packet passes through a GATEWAY, the GATEWAY observes the destination network ID to determine how to route the packet. If the destination network is connected to the GATEWAY, the lower 16 bits of the TCP address are used to produce a local TCP address in the destination network. If the destination network is not connected to the GATEWAY, the upper 8 bits are used to select a subsequent GATEWAY. We make no effort to specify how each individual network shall associate the internetwork TCP identifier with its local TCP address. We also do not rule out the possibility that the local network understands the internetwork addressing scheme and thus alleviates the GATEWAY of the routing responsibility.

PORT ADDRESSING

A receiving TCP is faced with the task of demultiplexing the stream of internetwork packets it receives and reconstructing the original messages for each destination process. Each operating system has its own internal means of identifying processes and ports. We assume that 16 bits are sufficient to serve as internetwork port identifiers. A sending process need not know how the destination port identification will be used. The destination TCP will be able to parse this number appropriately to find the proper buffer into which it will place arriving packets. We permit a large port number field to support processes which want to distinguish between many different messages streams concurrently. In reality, we do not care how the 16 bits are sliced up by the TCP's involved.

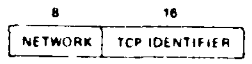


Fig. 4. TCP address.

Even though the transmitted port name field is large, it is still a compact external name for the internal representation of the port. The use of short names for port identifiers is often desirable to reduce transmission overhead and possibly reduce packet processing time at the destination TCP. Assigning short names to each port, however, requires an initial negotiation between source and destination to agree on a suitable short name assignment, the subsequent maintenance of conversion tables at both the source and the destination, and a final transaction to release the short name. For dynamic assignment of port names, this negotiation is generally necessary in any case.

SEGMENT AND PACKET FORMATS

As shown in Fig. 5, messages are broken by the TCP into segments whose format is shown in more detail in Fig. 6. The field lengths illustrated are merely suggestive. The first two fields (source port and destination port in the figure) have already been discussed in the preceding section on addressing. The uses of the third and fourth fields (window and acknowledgment in the figure) will be discussed later in the section on retransmission and duplicate detection.

We recall from Fig. 3 that an internetwork header contains both a sequence number and a byte count, as well as a flag field and a check sum. The uses of these fields are explained in the following section.

REASSEMBLY AND SEQUENCING

The reconstruction of a message at the receiving TCP clearly requires[1] that each internetwork packet carry a sequence number which is unique to its particular destination port message stream. The sequence numbers must be monotonic increasing (or decreasing) since they are used to reorder and reassemble arriving packets into a message. If the space of sequence numbers were infinite, we could simply assign the next one to each new packet. Clearly, this space cannot be infinite, and we will consider what problems a finite sequence number space will cause when we discuss retransmission and duplicate detection in the next section. We propose the following scheme for performing the sequencing of packets and hence the reconstruction of messages by the destination TCP.

A pair of ports will exchange one or more messages over a period of time. We could view the sequence of messages produced by one port as if it were embedded in an infinitely long stream of bytes. Each byte of the message has a unique sequence number which we take to be its byte location relative to the beginning of the stream. When a

[1] In the case of encrypted packets, a preliminary stage of reassembly may be required prior to decryption.

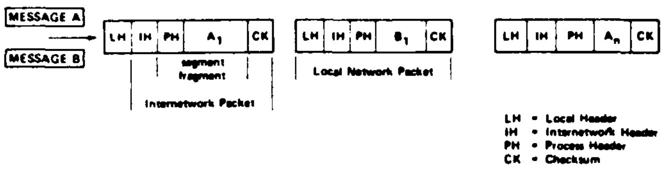


Fig. 5. Creation of segments and packets from messages.

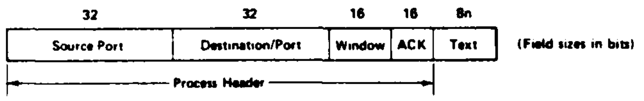


Fig. 6. Segment format (process header and text).

segment is extracted from the message by the source TCP and formatted for internetwork transmission, the relative location of the first byte of segment text is used as the sequence number for the packet. The byte count field in the internetwork header accounts for all the text in the segment (but does not include the check-sum bytes or the bytes in either internetwork or process header). We emphasize that the sequence number associated with a given packet is unique only to the pair of ports that are communicating (see Fig. 7). Arriving packets are examined to determine for which port they are intended. The sequence numbers on each arriving packet are then used to determine the relative location of the packet text in the messages under reconstruction. We note that this allows the exact position of the data in the reconstructed message to be determined even when pieces are still missing.

Every segment produced by a source TCP is packaged in a single internetwork packet and a check sum is computed over the text and process header associated with the segment.

The splitting of messages into segments by the TCP and the potential splitting of segments into smaller pieces by GATEWAYS creates the necessity for indicating to the destination TCP when the end of a segment (ES) has arrived and when the end of a message (EM) has arrived. The flag field of the internetwork header is used for this purpose (see Fig. 8).

The ES flag is set by the source TCP each time it prepares a segment for transmission. If it should happen that the message is completely contained in the segment, then the EM flag would also be set. The EM flag is also set on the last segment of a message, if the message could not be contained in one segment. These two flags are used by the destination TCP, respectively, to discover the presence of a check sum for a given segment and to discover that a complete message has arrived.

The ES and EM flags in the internetwork header are known to the GATEWAY and are of special importance when packets must be split apart for propagation through the next local network. We illustrate their use with an example in Fig. 9.

The original message A in Fig. 9 is shown split into two segments A_1 and A_2 and formatted by the TCP into a pair

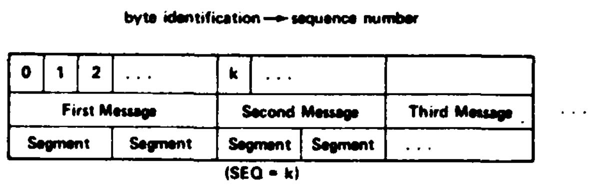

byte identification → sequence number
0 1 2 . . . k . . .
First Message
Second Message
Third Message
. . .
Segment Segment Segment Segment . . .
(SEQ = k)

Fig. 7. Assignment of sequence numbers.

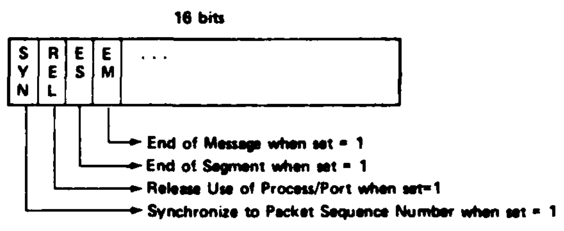

16 bits
SYN
REL
ES
EM
. . .
End of Message when set = 1
End of Segment when set = 1
Release Use of Process/Port when set=1
Synchronize to Packet Sequence Number when set = 1

Fig. 8. Internetwork header flag field.

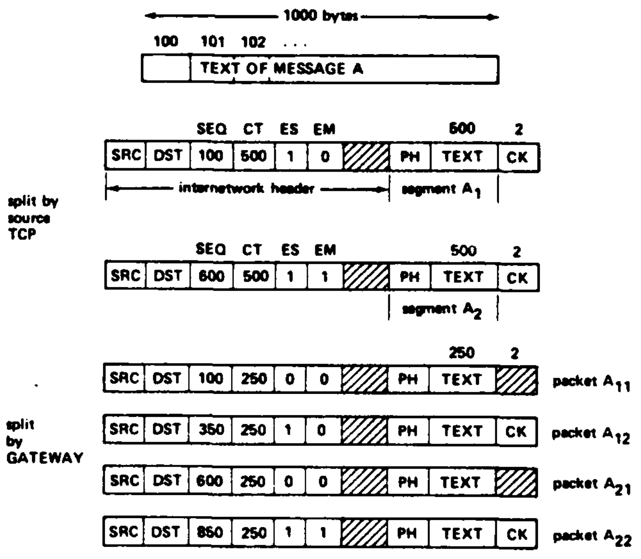


Fig. 9. Message splitting and packet splitting.

of internetwork packets. Packets A_1 and A_2 have their ES bits set, and A_2 has its EM bit set as well. When packet A_1 passes through the GATEWAY, it is split into two pieces: packet A_{11} for which neither EM nor ES bits are set, and packet A_{12} whose ES bit is set. Similarly, packet A_2 is split such that the first piece, packet A_{21}, has neither bit set, but packet A_{22} has both bits set. The sequence number field (SEQ) and the byte count field (CT) of each packet is modified by the GATEWAY to properly identify the text bytes of each packet. The GATEWAY need only examine the internetwork header to do fragmentation.

The destination TCP, upon reassembling segment A_1, will detect the ES flag and will verify the check sum it knows is contained in packet A_{12}. Upon receipt of packet A_{22}, assuming all other packets have arrived, the destination TCP detects that it has reassembled a complete message and can now advise the destination process of its receipt.

RETRANSMISSION AND DUPLICATE DETECTION

No transmission can be 100 percent reliable. We propose a timeout and positive acknowledgment mechanism which will allow TCP's to recover from packet losses from one HOST to another. A TCP transmits packets and waits for replies (acknowledgements) that are carried in the reverse packet stream. If no acknowledgment for a particular packet is received, the TCP will retransmit. It is our expectation that the HOST level retransmission mechanism, which is described in the following paragraphs, will not be called upon very often in practice. Evidence already exists[2] that individual networks can be effectively constructed without this feature. However, the inclusion of a HOST retransmission capability makes it possible to recover from occasional network problems and allows a wide range of HOST protocol strategies to be incorporated. We envision it will occasionally be invoked to allow HOST accommodation to infrequent overdemands for limited buffer resources, and otherwise not used much.

Any retransmission policy requires some means by which the receiver can detect duplicate arrivals. Even if an infinite number of distinct packet sequence numbers were available, the receiver would still have the problem of knowing how long to remember previously received packets in order to detect duplicates. Matters are complicated by the fact that only a finite number of distinct sequence numbers are in fact available, and if they are reused, the receiver must be able to distinguish between new transmissions and retransmissions.

A *window* strategy, similar to that used by the French CYCLADES system (voie virtuelle transmission mode [8]) and the ARPANET very distant HOST connection [18], is proposed here (see Fig. 10).

Suppose that the sequence number field in the internetwork header permits sequence numbers to range from 0 to $n - 1$. We assume that the sender will not transmit more than w bytes without receiving an acknowledgment. The w bytes serve as the window (see Fig. 11). Clearly, w must be less than n. The rules for sender and receiver are as follows.

Sender: Let L be the sequence number associated with the left window edge.

1) The sender transmits bytes from segments whose text lies between L and up to $L + w - 1$.

2) On timeout (duration unspecified), the sender retransmits unacknowledged bytes.

3) On receipt of acknowledgment consisting of the receiver's current left window edge, the sender's left window edge is advanced over the acknowledged bytes (advancing the right window edge implicitly).

Receiver:

1) Arriving packets whose sequence numbers coincide with the receiver's current left window edge are acknowledged by sending to the source the next sequence number expected. This effectively acknowledges bytes in between. The left window edge is advanced to the next sequence number expected.

2) Packets arriving with a sequence number to the left of the window edge (or, in fact, outside of the window) are discarded, and the current left window edge is returned as acknowledgment.

3) Packets whose sequence numbers lie within the receiver's window but do not coincide with the receiver's left window edge are optionally kept or discarded, but are not acknowledged. This is the case when packets arrive out of order.

We make some observations on this strategy. First, all computations with sequence numbers and window edges must be made modulo n (e.g., byte 0 follows byte $n - 1$). Second, w must be less than $n/2$[3]; otherwise a retransmission may appear to the receiver to be a new transmission in the case that the receiver has accepted a window's worth of incoming packets, but all acknowledgments have been lost. Third, the receiver can either save or discard arriving packets whose sequence numbers do not coincide with the receiver's left window. Thus, in the simplest implementation, the receiver need not buffer more than one packet per message stream if space is critical. Fourth, multiple packets can be acknowledged simultaneously. Fifth, the receiver is able to deliver messages to processes in their proper order as a natural result of the reassembly mechanism. Sixth, when duplicates are detected, the acknowledgment method used naturally works to resynchronize sender and receiver. Furthermore, if the receiver accepts packets whose sequence numbers lie within the current window but

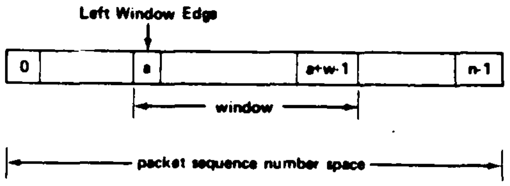


Fig. 10. The window concept.

1	Source Address	
2	Destination Address	
3	Next Packet Seq.	
4	Current Buffer Size	
5	Next Write Position	
6	Next Read Position	
7	End Read Position	
8	No. Retrans.	Max Retrans.
9	Timeout	Flags
10	Curr. Ack	Window

Fig. 11. Conceptual TCB format.

[2] The ARPANET is one such example.

[3] Actually $n/2$ is merely a convenient number to use; it is only required that a retransmission not appear to be a new transmission.

which are not coincident with the left window edge, an acknowledgment consisting of the current left window edge would act as a stimulus to cause retransmission of the unacknowledged bytes. Finally, we mention an overlap problem which results from retransmission, packet splitting, and alternate routing of packets through different GATEWAYS.

A 600-byte packet might pass through one GATEWAY and be broken into two 300-byte packets. On retransmission, the same packet might be broken into three 200-byte packets going through a different GATEWAY. Since each byte has a sequence number, there is no confusion at the receiving TCP. We leave for later the issue of initially synchronizing the sender and receiver left window edges and the window size.

FLOW CONTROL

Every segment that arrives at the destination TCP is ultimately acknowledged by returning the sequence number of the next segment which must be passed to the process (it may not yet have arrived).

Earlier we described the use of a sequence number space and window to aid in duplicate detection. Acknowledgments are carried in the process header (see Fig. 6) and along with them there is provision for a "suggested window" which the receiver can use to control the flow of data from the sender. This is intended to be the main component of the process flow control mechanism. The receiver is free to vary the window size according to any algorithm it desires so long as the window size never exceeds half the sequence number space.[3]

This flow control mechanism is exceedingly powerful and flexible and does not suffer from synchronization troubles that may be encountered by incremental buffer allocation schemes [9],[10]. However, it relies heavily on an effective retransmission strategy. The receiver can reduce the window even while packets are en route from the sender whose window is presently larger. The net effect of this reduction will be that the receiver may discard incoming packets (they may be outside the window) and reiterate the current window size along with a current window edge as acknowledgment. By the same token, the sender can, upon occasion, choose to send more than a window's worth of data on the possibility that the receiver will expand the window to accept it (of course, the sender must not send more than half the sequence number space at any time). Normally, we would expect the sender to abide by the window limitation. Expansion of the window by the receiver merely allows more data to be accepted. For the receiving HOST with a small amount of buffer space, a strategy of discarding all packets whose sequence numbers do not coincide with the current left edge of the window is probably necessary, but it will incur the expense of extra delay and overhead for retransmission.

TCP INPUT/OUTPUT HANDLING

The TCP has a component which handles input/output (I/O) to and from the network.[4] When a packet has arrived, it validates the addresses and places the packet on a queue. A pool of buffers can be set up to handle arrivals, and if all available buffers are used up, succeeding arrivals can be discarded since unacknowledged packets will be retransmitted.

On output, a smaller amount of buffering is needed, since process buffers can hold the data to be transmitted. Perhaps double buffering will be adequate. We make no attempt to specify how the buffering should be done, except to require that it be able to service the network with as little overhead as possible. Packet sized buffers, one or more ring buffers, or any other combination are possible candidates.

When a packet arrives at the destination TCP, it is placed on a queue which the TCP services frequently. For example, the TCP could be interrupted when a queue placement occurs. The TCP then attempts to place the packet text into the proper place in the appropriate process receive buffer. If the packet terminates a segment, then it can be checksummed and acknowledged. Placement may fail for several reasons.

1) The destination process may not be prepared to receive from the stated source, or the destination port ID may not exist.

2) There may be insufficient buffer space for the text.

3) The beginning sequence number of the text may not coincide with the next sequence number to be delivered to the process (e.g., the packet has arrived out of order).

In the first case, the TCP should simply discard the packet (thus far, no provision has been made for error acknowledgments). In the second and third cases, the packet sequence number can be inspected to determine whether the packet text lies within the legitimate window for reception. If it does, the TCP may optionally keep the packet queued for later processing. If not, the TCP can discard the packet. In either case the TCP can optionally acknowledge with the current left window edge.

It may happen that the process receive buffer is not present in the active memory of the HOST, but is stored on secondary storage. If this is the case, the TCP can prompt the scheduler to bring in the appropriate buffer and the packet can be queued for later processing.

If there are no more input buffers available to the TCP for temporary queueing of incoming packets, and if the TCP cannot quickly use the arriving data (e.g., a TCP to TCP message), then the packet is discarded. Assuming a sensibly functioning system, no other processes than the one for which the packet was intended should be affected by this discarding. If the delayed processing queue grows

[4] This component can serve to handle other protocols whose associated control programs are designated by internetwork destination address.

excessively long, any packets in it can be safely discarded since none of them have yet been acknowledged. Congestion at the TCP level is flexibly handled owing to the robust retransmission and duplicate detection strategy.

TCP/PROCESS COMMUNICATION

In order to send a message, a process sets up its text in a buffer region in its own address space, inserts the requisite control information (described in the following list) in a transmit control block (TCB) and passes control to the TCP. The exact form of a TCB is not specified here, but it might take the form of a passed pointer, a pseudointerrupt, or various other forms. To receive a message in its address space, a process sets up a receive buffer, inserts the requisite control information in a receive control block (RCB) and again passes control to the TCP.

In some simple systems, the buffer space may in fact be provided by the TCP. For simplicity we assume that a ring buffer is used by each process, but other structures (e.g., buffer chaining) are not ruled out.

A possible format for the TCB is shown in Fig. 11. The TCB contains information necessary to allow the TCP to extract and send the process data. Some of the information might be implicitly known, but we are not concerned with that level of detail. The various fields in the TCB are described as follows.

1) *Source Address*: This is the full net/HOST/TCP/port address of the transmitter.

2) *Destination Address*: This is the full net/HOST/TCP/port of the receiver.

3) *Next Packet Sequence Number*: This is the sequence number to be used for the next packet the TCP will transmit from this port.

4) *Current Buffer Size*: This is the present size of the process transmit buffer.

5) *Next Write Position*: This is the address of the next position in the buffer at which the process can place new data for transmission.

6) *Next Read Position*: This is the address at which the TCP should begin reading to build the next segment for output.

7) *End Read Position*: This is the address at which the TCP should halt transmission. Initially 6) and 7) bound the message which the process wishes to transmit.

8) *Number of Retransmissions/Maximum Retransmissions*: These fields enable the TCP to keep track of the number of times it has retransmitted the data and could be omitted if the TCP is not to give up.

9) *Timeout/Flags*: The timeout field specifies the delay after which unacknowledged data should be retransmitted. The flag field is used for semaphores and other TCP/process synchronization, status reporting, etc.

10) *Current Acknowledgment/Window*: The current acknowledgment field identifies the first byte of data still unacknowledged by the destination TCP.

The read and write positions move circularly around the transmit buffer, with the write position always to the left (module the buffer size) of the read position.

The next packet sequence number should be constrained to be less than or equal to the sum of the current acknowledgment and the window fields. In any event, the next sequence number should not exceed the sum of the current acknowledgment and half of the maximum possible sequence number (to avoid confusing the receiver's duplicate detection algorithm). A possible buffer layout is shown in Fig. 12.

The RCB is substantially the same, except that the end read field is replaced by a partial segment check-sum register which permits the receiving TCP to compute and remember partial check sums in the event that a segment arrives in several packets. When the final packet of the segment arrives, the TCP can verify the check sum and if successful, acknowledge the segment.

CONNECTIONS AND ASSOCIATIONS

Much of the thinking about process-to-process communication in packet switched networks has been influenced by the ubiquitous telephone system. The HOST-HOST protocol for the ARPANET deals explicitly with the opening and closing of simplex connections between processes [9],[10]. Evidence has been presented that message-based "connection-free" protocols can be constructed [12], and this leads us to carefully examine the notion of a connection.

The term *connection* has a wide variety of meanings. It can refer to a physical or logical path between two entities, it can refer to the flow over the path, it can inferentially refer to an action associated with the setting up of a path, or it can refer to an association between two or more entities, with or without regard to any path between them. In this paper, we do not explicitly reject the term connection, since it is in such widespread use, and does connote a meaningful relation, but consider it exclusively in the sense of an association between two or more entities without regard to a path. To be more precise about our intent, we shall define the relationship between two or more ports that are in communication, or are prepared to communicate to be an *association*. Ports that are associated with each other are called *associates*.

It is clear that for any communication to take place between two processes, one must be able to address the other. The two important cases here are that the destination port may have a global and unchanging address or that it may be globally unique but dynamically reassigned. While in either case the sender may have to learn the destination address, given the destination name, only in the second instance is there a requirement for learning the address from the destination (or its representative) each time an association is desired. Only after the source has learned how to address the destination can an association be said to have occurred. But this is not yet sufficient. If

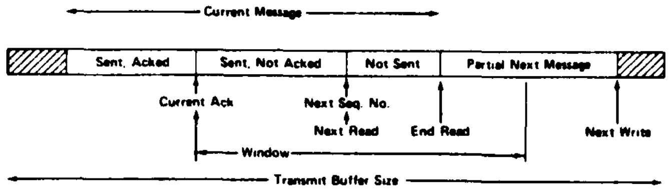


Fig. 12. Transmit buffer layout.

ordering of delivered messages is also desired, both TCP's must maintain sufficient information to allow proper sequencing. When this information is also present at both ends, then an association is said to have occurred.

Note that we have not said anything about a path, nor anything which implies that either end be aware of the condition of the other. Only when both partners are prepared to communicate with each other has an association occurred, and it is possible that neither partner may be able to verify that an association exists until some data flows between them.

CONNECTION-FREE PROTOCOLS WITH ASSOCIATIONS

In the ARPANET, the interface message processors (IMP's) do not have to open and close connections from source to destination. The reason for this is that connections are, in effect, always open, since the address of every source and destination is never[5] reassigned. When the name and the place are static and unchanging, it is only necessary to *label a packet with source and destination* to transmit it through the network. In our parlance, every source and destination forms an association.

In the case of processes, however, we find that port addresses are continually being used and reused. Some ever-present processes could be assigned fixed addresses which do not change (e.g., the logger process). If we supposed, however, that every TCP had an infinite supply of port addresses so that no old address would ever be reused, then any dynamically created port would be assigned the next unused address. In such an environment, there could never be any confusion by source and destination TCP as to the intended recipient or implied source of each message, and all ports would be associates.

Unfortunately, TCP's (or more properly, operating systems) tend not to have an infinite supply of internal port addresses. These internal addresses are reassigned after the demise of each port. Walden [12] suggests that a set of unique uniform external port addresses could be supplied by a central registry. A newly created port could apply to the central registry for an address which the central registry would guarantee to be unused by any HOST system in the network. Each TCP could maintain tables matching external names with internal ones, and use the external ones for communication with other processes. This idea violates the premise that interprocess communication should not require centralized control. One would have to extend the central registry service to include all HOST's in all the interconnected networks to apply this idea to our situation, and we therefore do not attempt to adopt it.

[5] Unless the IMP is physically moved to another site, or the HOST is connected to a different IMP.

Let us consider the situation from the standpoint of the TCP. In order to send or receive data for a given port, the TCP needs to set up a TCB and RCB and initialize the window size and left window edge for both. On the receive side, this task might even be delayed until the first packet destined for a given port arrives. By convention, the first packet should be marked so that the receiver will synchronize to the received sequence number.

On the send side, the first request to transmit could cause a TCB to be set up with some initial sequence number (say, zero) and an assumed window size. The receiving TCP can reject the packet if it wishes and notify the sending TCP of the correct window size via the acknowledgment mechanism, but only if either

1) we insist that the first packet be a complete segment;
2) an acknowledgment can be sent for the first packet (even if not a segment, as long as the acknowledgment specifies the next sequence number such that the source also understands that no bytes have been accepted).

It is apparent, therefore, that the synchronizing of window size and left window edge can be accomplished without what would ordinarily be called a connection setup.

The first packet referencing a newly created RCB sent from one associate to another can be marked with a bit which requests that the receiver synchronize his left window edge with the sequence number of the arriving packet (see SYN bit in Fig. 8). The TCP can examine the source and destination port addresses in the packet and in the RCB to decide whether to accept or ignore the request.

Provision should be made for a destination process to specify that it is willing to LISTEN to a specific port or "any" port. This last idea permits processes such as the logger process to accept data arriving from unspecified sources. This is purely a HOST matter, however.

The initial packet may contain data which can be stored or discarded by the destination, depending on the availability of destination buffer space at the time. In the other direction, acknowledgment is returned for receipt of data which also specifies the receiver's window size.

If the receiving TCP should want to reject the synchronization request, it merely transmits an acknowledgment carrying a release (REL) bit (see Fig. 8) indicating that the destination port address is unknown or inaccessible. The sending HOST waits for the acknowledgment (after accepting or rejecting the synchronization request) before sending the next message or segment. This rejection is quite different from a negative data acknowledgment. We do not have explicit negative acknowledgments. If no acknowledgment is returned, the sending HOST may retransmit without introducing confusion if, for example, the left window edge is not changed on the retransmission.

Because messages may be broken up into many packets for transmission or during transmission, it will be necessary to ignore the REL flag except in the case that the EM flag is also set. This could be accomplished either by the TCP or by the GATEWAY which could reset the flag on all but the packet containing the set EM flag (see Fig. 9).

At the end of an association, the TCP sends a packet with ES, EM, and REL flags set. The packet sequence number scheme will alert the receiving TCP if there are still outstanding packets in transit which have not yet arrived, so a premature dissociation cannot occur.

To assure that both TCP's are aware that the association has ended, we insist that the receiving TCP respond to the REL by sending a REL acknowledgment of its own.

Suppose now that a process sends a single message to an associate including an REL along with the data. Assuming an RCB has been prepared for the receiving TCP to accept the data, the TCP will accumulate the incoming packets until the one marked ES, EM, REL arrives, at which point a REL is returned to the sender. The association is thereby terminated and the appropriate TCB and RCB are destroyed. If the first packet of a message contains a SYN request bit and the last packet contains ES, EM, and REL bits, then data will flow "one message at a time." This mode is very similar to the scheme described by Walden [12], since each succeeding message can only be accepted at the receiver after a new LISTEN (like Walden's RECEIVE) command is issued by the receiving process to its serving TCP. Note that only if the acknowledgment is received by the sender can the association be terminated properly. It has been pointed out[6] that the receiver may erroneously accept duplicate transmissions if the sender does not receive the acknowledgment. This may happen if the sender transmits a duplicate message with the SYN and REL bits set and the destination has already destroyed any record of the previous transmission. One way of preventing this problem is to destroy the record of the association at the destination only after some known and suitably chosen timeout. However, this implies that a new association with the same source and destination port identifiers could not be established until this timeout had expired. This problem can occur even with sequences of messages whose SYN and REL bits are separated into different internetwork packets. We recognize that this problem must be solved, but do not go into further detail here.

Alternatively, both processes can send one message, causing the respective TCP's to allocate RCB/TCB pairs at both ends which rendezvous with the exchanged data and then disappear. If the overhead of creating and destroying RCB's and TCB's is small, such a protocol might be adequate for most low-bandwidth uses. This idea might also form the basis for a relatively secure transmission system. If the communicating processes agree to change their external port addresses in some way known only to each other (i.e., pseudorandom), then each message will appear to the outside world as if it is part of a different association message stream. Even if the data is intercepted by a third party, he will have no way of knowing that the data should in fact be considered part of a sequence of messages.

We have described the way in which processes develop associations with each other, thereby becoming associates for possible exchange of data. These associations need not involve the transmission of data prior to their formation and indeed two associates need not be able to determine that they are associates until they attempt to communicate.

[6] S. Crocker of ARPA/IPT.

CONCLUSIONS

We have discussed some fundamental issues related to the interconnection of packet switching networks. In particular, we have described a simple but very powerful and flexible protocol which provides for variation in individual network packet sizes, transmission failures, sequencing, flow control, and the creation and destruction of process-to-process associations. We have considered some of the implementation issues that arise and found that the proposed protocol is implementable by HOST's of widely varying capacity.

The next important step is to produce a detailed specification of the protocol so that some initial experiments with it can be performed. These experiments are needed to determine some of the operational parameters (e.g., how often and how far out of order do packets actually arrive; what sort of delay is there between segment acknowledgments; what should be retransmission timeouts be?) of the proposed protocol.

ACKNOWLEDGMENT

The authors wish to thank a number of colleagues for helpful comments during early discussions of international network protocols, especially R. Metcalfe, R. Scantlebury, D. Walden, and H. Zimmerman; D. Davies and L. Pouzin who constructively commented on the fragmentation and accounting issues; and S. Crocker who commented on the creation and destruction of associations.

REFERENCES

[1] L. Roberts and B. Wessler, "Computer network development to achieve resource sharing," in *1970 Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 543–549.

[2] L. Pouzin, "Presentation and major design aspects of the CYCLADES computer network," in *Proc. 3rd Data Communications Symp.*, 1973.

[3] F. R. E. Dell, "Features of a proposed synchronous data network," in *Proc. 2nd Symp. Problems in the Optimization of Data Communications Systems*, 1971, pp. 50–57.

[4] R. A. Scantlebury and P. T. Wilkinson, "The design of a switching system to allow remote access to computer services by other computers and terminal devices," in *Proc. 2nd Symp. Problems in the Optimization of Data Communications Systems*, 1971, pp. 160–167.
[5] D. L. A. Barber, "The European computer network project," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 192–200.
[6] R. Despres, "A packet switching network with graceful saturated operation," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 345–351.
[7] R. E. Kahn and W. R. Crowther, "Flow control in a resource-sharing computer network," *IEEE Trans. Commun.*, vol. COM-20, pp. 539–546, June 1972.
[8] J. F. Chambon, M. Elie, J. Le Bihan, G. LeLann, and H. Zimmerman, "Functional specification of transmission station in the CYCLADES network. ST-ST protocol" (in French), I.R.I.A. Tech. Rep. SCH502.3, May 1973.
[9] S. Carr, S. Crocker, and V. Cerf, "HOST-HOST Communication Protocol In the ARPA Network," in *Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 589–597.
[10] A. McKenzie, "HOST/HOST protocol for the ARPA network," in *Current Network Protocols*, Network Information Cen., Menlo Park, Calif., NIC 8246, Jan. 1972.
[11] L. Pouzin, "Address format in Mitranet," NIC 14497, INWG 20, Jan. 1973.
[12] D. Walden, "A system for interprocess communication in a resource sharing computer network," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 221–230, Apr. 1972.
[13] B. Lampson, "A scheduling philosophy for multiprocessing systems," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 347–360, May 1968.
[14] F. E. Heart, R. E. Kahn, S. Ornstein, W. Crowther, and D. Walden, "The interface message processor for the ARPA computer network," in *Proc. Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 551–567.
[15] N. G. Anslow and J. Hanscoff, "Implementation of international data exchange networks," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 181–184
[16] A. McKenzie, "HOST/HOST protocol design considerations," INWG Note 16, NIC 13879, Jan. 1973.
[17] R. E. Kahn, "Resource-sharing computer communication networks," *Proc. IEEE*, vol. 60, pp. 1397–1407, Nov. 1972.
[18] Bolt, Beranek, and Newman, "Specification for the interconnection of a host and an IMP," Bolt Beranek and Newman, Inc., Cambridge, Mass., BBN Rep. 1822 (revised), Apr. 1973.

Vinton G. Cerf was born in New Haven, Conn., in 1943. He did undergraduate work in mathematics at Stanford University, Stanford, Calif., and received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, Calif., in 1972.

He was with IBM in Los Angeles from 1965 through 1967 and consulted and/or worked part time at UCLA from 1967 through 1972. Currently he is Assistant Professor of Computer Science and Electrical Engineering at Stanford University, and consultant to Cabledata Associates. Most of his current research is supported by the Defense Advanced Research Projects Agency and by the National Science Foundation on the technology and economics of computer networking. He is Chairman of IFIP TC6.1, an international network working group which is studying the problem of packet network interconnection.

★

Robert E. Kahn (M'65) was born in Brooklyn, N. Y., on December 23, 1938. He received the B.E.E. degree from the City College of New York, New York, in 1960, and the M.A. and Ph.D. degrees from Princeton University, Princeton, N. J., in 1962 and 1964, respectively.

From 1960 to 1962 he was a Member of the Technical Staff of Bell Telephone Laboratories, Murray Hill, N. J., engaged in traffic and communication studies. From 1964 to 1966 he was a Ford Postdoctoral Fellow and an Assistant Professor of Electrical Engineering at the Massachusetts Institute of Technology, Cambridge, where he worked on communications and information theory. From 1966 to 1972 he was a Senior Scientist at Bolt Beranek and Newman, Inc., Cambridge, Mass., where he worked on computer communications network design and techniques for distributed computation. Since 1972 he has been with the Advanced Research Projects Agency, Department of Defense, Arlington, Va.

Dr. Kahn is a member of Tau Beta Pi, Sigma Xi, Eta Kappa Nu, the Institute of Mathematical Statistics, and the Mathematical Association of America. He was selected to serve as a National Lecturer for the Association for Computing Machinery in 1972.

Reprinted by permission from
IEEE TRANSACTIONS ON COMMUNICATIONS
Vol. COM-22, No. 5, May 1974

[29] L. G. Roberts and B. D. Wessler, "Computer network development to achieve resource sharing," in *AFIPS SJCC Proc.*, vol. 36, p. 543, May 1970.
[30] M. Shaw, W. A. Wulf and R. L. London, "Abstraction and verification in Alphard: Defining and specifying iteration and generators," *CACM*, vol. 20, no. 8, p. 553, Aug. 1977.
[31] R. F. Sproull, "Omnigraph-Simple terminal-independent graphics software," Xerox Palo Alto Res. Center, CSL-73-4, 1973.
[32] ——, "InterLisp display primitives," Xerox Palo Alto Res. Center, 1977, informal note.
[33] R. F. Sproull and E. L. Thomas, "A network graphics protocol," *Comput. Graphics*, vol. 8, no. 3, Fall 1974.
[34] C. A. Sunshine, "Survey of protocol definition and verification techniques," in *Computer Network Protocols*, A. Danthine, Ed. Liege, Belgium, Feb. 1978.
[35] W. Teitelman, "A display oriented programmer's assistant," Xerox Palo Alto Res. Center, CSL-77-3, 1977.
[36] R. H. Thomas, "A resource sharing executive for the ARPAnet," in *AFIPS Proc.*, NCC, p. 155, 1973.
[37] ——, "MSG: The interprocess communication facility for the national software works," Bolt Beranek and Newman, Rep. 3483.
[38] D. C. Walden, "A system for interprocess communication in a resource-sharing computer network," *CACM*, vol. 15, no. 4, p. 221, Apr. 1972.
[39] J. E. White, "A high-level framework for network-based resource sharing," *AFIPS Proc.*, NCC, p. 561, 1976.
[40] P. A. Woodsford, "The design and implementation of the GINO 3D graphics software package," *Software Practice and Experience*, vol. 1, p. 335, Oct. 1971.

Issues in Packet-Network Interconnection

VINTON G. CERF AND PETER T. KIRSTEIN

Invited Paper

Abstract—This paper introduces the wide range of technical, legal, and political issues associated with the interconnection of packet-switched data communication networks. Motivations for interconnection are given, desired user services are described, and a range of technical choices for achieving interconnection are compared. Issues such as the level of interconnection, the role of gateways, naming and addressing, flow and congestion control, accounting and access control, and basic internet services are discussed in detail. The CCITT X.25/X.75 packet-network interface recommendations are evaluated in terms of their applicability to network interconnection. Alternatives such as datagram operation and general host gateways are compared with the virtual circuit methods. Some observations on the regulatory aspects of interconnection are offered and the paper concludes with a statement of open research problems and some tentative conclusions.

I. Introduction

IT IS THE THEME of many papers in this issue, that people need access to data resources. In many cases this access must be over large distances, in others it may be local to a building or a single site. Data networks have been set up to meet many user needs—often, but not necessarily, using packet-switching technology. For single organizations, these data networks are often private ones, built with a technology optimized to the specific application. For communication between organizations, these networks are being set up by licensed carriers. In North America, there are many such licensed carriers, e.g., TELENET [1], DATAPAC [2], and TYMNET [3]. In the rest of the world, the Post, Telegraph, and Telephone Authority (PTT) in each country has a near monopoly on such services; special public data networks being set up in these countries include TRANSPAC [5] in France, EURONET [6] for inter-European traffic, DDX [7] in Japan, EDS [8] in the Federal Republic of Germany, and the Nordic Public Data Network (NPDN, [9]) in Scandinavia. These public data networks are considered in greater detail in other references (e.g., [10]-[12]). Most of the above networks use packet-switching technology; some of them, e.g., EDS and the NPDN, do not do so yet, but may do so in the future. In some cases special data networks have been authorized for specific communities, e.g., SITA [13] for the airlines, and SWIFT [14] for the banks. In addition many private networks have been set up among individual organizations, and experimental networks of different technologies have been developed also, e.g., ARPANET [15], [16], CYCLADES [17], ETHERNET [18], SPYDER [19], PRNET [20], [21] and SATNET [22].

Manuscript received June 20, 1978; revised July 21, 1978.
V. G. Cerf is with the Advanced Research Projects Agency, U.S. Department of Defense, Arlington, VA 22209.
P. T. Kirstein is with the Department of Statistic and Computer Science, University College, London, England.

It is a common user requirement that a single terminal and access port should be able to access any computing resource the user may desire–even if the resource is on another data network. From this requirement, there is a clear user need to have data networks connected together. By the same token, the providers of data network services would like to have their networks used as intensively as possible; thus they also have a strong motivation to connect their data networks to others. As a result of these considerations, there has been a high recent interest in the issues arising in the connection of data networks [23]-[26], [32].

From the user viewpoint, the requirement for interconnection of data networks is independent of the network technology. From the implementation viewpoint, there can be some considerable complications in connecting networks of widely different technologies–such as circuit-switched and datagram packet-switched networks (these terms are explained below). On the whole we will consider only, in this paper, the interconnection of packet-switched data networks. In many cases, however, the arguments will be equally valid for the interconnection of packet-switched to circuit-switched networks.

Network interconnection raises a great many technical, legal, and political questions and issues. The technical issues generally revolve around mechanisms for achieving interconnection and their performance. How can networks be interconnected so that packets can flow in a controllable way from one net to another? Should all computer systems on all nets be able to communicate with each other? How can this be achieved? What kind of performance can be achieved with a set of interconnected networks of widely varying internal design and operating characteristics? How are terminals to be given access to resources in other networks? What protocols are required to achieve this? Should the protocols of one net be translated into those of another, or should common protocols be defined? What kinds of communication protocol standards are needed to support efficient and useful interconnection? Who should take responsibility for setting standards?

The legal and political issues are at least as complex as the technical ones. Can private networks interconnect to each other or must they do so through the mediation of a public network? How is privacy to be protected? Should there be control over the kinds of data which move from one net to another? Are there international agreements and conventions which might be affected by international interconnection of data networks? What kinds of charging and accounting policies should apply to multinetwork traffic? How can faults and errors be diagnosed in a multinet environment? Who should be responsible for correcting such faults? Who should be responsible for maintaining the gateways which connect nets together?

We cannot possibly answer all of these questions in this paper, but we deal with many of them in the sections below.

This paper is divided into eleven sections. In the next section we provide some definitions, and in Section III we explore some of the motivations for network interconnection. In Section IV we discuss the range of end-user service requirements and choices for providing multinetwork service. Section V reviews the concept of computer-communication protocol layering. Section VI reviews the basic interconnection choices and introduces the concept of gateways between nets, protocol translation and the impact of common protocols; it elaborates also on the function of gateways. Section VII discusses the CCITT recommendations X.25 and X.75 and their role in network interconnection. Section VIII describes some of the network interconnections achieved and some of the experiments in progress. Section IX outlines regulatory issues raised by network interconnection alternatives. Section X mentions some unresolved research questions, and the final section offers some tentative conclusions on network interconnection issues.

II. The Definition of Terms

The vocabulary of networking is extensive and not always consistent. We introduce some generic terms below which we will use in this paper for purposes of discussion. It is important for the reader not to make any *a priori* assumptions about the physical realization of the objects named or of the boundary of jurisdictions owning or managing them. For instance, a gateway (see below) might be implemented to share the hardware of a packet switch and be owned by a packet-switching service carrier; alternatively it might be embedded in a host computer which subscribes to service on two or more computer networks. Roughly speaking, we are assigning names to groups of functions which may or may not be realized as physically distinct entities.

Packet: A packet of information is a finite sequence of bits, divided into a control header part and a data part. The header will contain enough information for the packet to be routed to its destination. There will usually be some checks on each such packet, so that any switch through which the packet passes may exercise error control. Packets are generally associated with internal packet-network operation and are not necessarily visible to host computers attached to the network.

Datagram: A finite length packet of data together with destination host address information (and, usually, source address) which can be exchanged in its entirety between hosts, independent of all other datagrams sent through a packet switched network. Typically, the maximum length of a datagram lies between 1000 and 8000 bits.

Gateway: The collection of hardware and software required to effect the interconnection of two or more data networks, enabling the passage of user data from one to another.

Host: The collection of hardware and software which utilizes the basic packet-switching service to support end-to-end interprocess communication and user services.

Packet Switch: The collection of hardware and software resources which implements all intranetwork procedures such as routing, resource allocation, and error control and provides access to network packet-switching services through a host/network interface.

Protocol: A set of communication conventions, including formats and procedures which allow two or more end points to communicate. The end points may be packet switches, hosts, terminals, people, file systems, etc.

Protocol Translator: A collection of software, and possibly hardware, required to convert the high level protocols used in one network to those used in another.

Terminal: A collection of hardware and possibly software which may be as simple as a character-mode teletype or as complex as a full scale computer system. As terminals increase in capability, the distinction between "host" and "terminal" may become a matter of nomenclature without technical substance.

Virtual Circuit: A logical channel between source and destination packet switches in a packet-switched network. A virtual circuit requires some form of "setup" which may or may not be visible to the subscriber. Packets sent on a virtual circuit are delivered in the order sent, but with varying delay.

PTT: Technically PTT stands for Post, Telegraph, and Telephone Authority; this authority has a different form in different countries. In this paper, by PTT we mean merely the authority (or authorities) licensed in each country to offer public data transmission services.

We have attempted to make these definitions as noncontroversial as possible. For example, in the definition of packet switch, we alluded to a host/network interface. The reader should not assume that subscriber services are limited to those offered through the host/network interface. The packet-switching carrier might also offer host-based services and terminal access mechanisms as additional subscriber services.

III. The Motivating Forces in the Interconnection of Data Networks

In the introduction, we mentioned that there was a strong interest, among both the users and suppliers of data serivces, in the interconnection of data networks. However, the technical interests of the different parties are not identical. The end user would merely like to be able to access any resources from a single terminal, with a single access port, as economically as possible according to his own performance criteria. A Public Carrier, or PTT, has a strong motivation to connect its network to other PTT's. As in the telephone system, the concept of all subscribers being accessible through a single Public Data Service, is considered highly desirable; however the different PTT's may have restricted geographic coverage, or only a specific market penetration.

The motivation of the PTT's to interface to private networks is weaker and more complex. They always provide facilities to attach single terminals, where a terminal may be a complex computer system; they are often not interested, at present, in making any special arrangements when the "terminal" is a whole computer network. The operators of private networks often have a vital interest in connecting their networks to other private networks and to the public ones. Even though in many cases the bulk of its traffic is internal to the private network, which is why it was set up in the first place, there is usually a vital need to access resources not available on that network. The regulatory limitations often imposed on the method of interconnection of private networks are discussed in Section IX. In some countries, it is not permitted to build private networks using leased line services, but intrabuilding networks may be permitted. Interconnection of such local networks to public networks may play a crucial role in making the local network useful.

To date the PTT's have tried to standardize on access procedures for their Public-Packet Data Services. The standardization has taken place in the International Consultative Committee on Telegraphy and Telephony (called CCITT) in a set of recommendations called X.3, X.25, X.28, and X.29 ([27]-[29]). Not all PTT's have such forms of access yet, but most of the industrialized nations in the West are moving in this direction. This series of recommendations is discussed in much more detail in Section VI; it does not pay special attention to the attachment of private networks ([31], [32]), but the recommendations are themselves expected to change to meet this requirement. The PTT's are agreeing on a set of interface recommendations and procedures called X.75 [33], to connect their networks to each other; so far this interface procedure (and its corresponding hardware) is not intended to be provided to private networks.

While most PTT's have preferred to ignore the technical implications of the attachment of private networks to the public ones, most private network operators cannot ignore this requirement. They are often motivated to add some extra "Foreign Exchange" capability as an afterthought, with minimum change to their intranetwork procedures; this approach can be successful up to a point, but will usually be limited by the lack of high-level procedures between the different networks. These high-level procedures have not yet been considered by CCITT, but it has been proposed that CCITT Study Group VII investigate high-level procedures and architectural models, in cooperation with the investigation of "open system architectures" by Technical Committee 97, Sub-Committee 16 of the International Standards Organisation (ISO). This subject is also considered later in this paper, in Section VI.

An aim of these standardization exercises is to ensure that both manufacturer and user implementations of network resources can communicate with each other through single private or public data networks. A consequence should be that the resources are also compatibly accessible over connected data networks.

Depending on the applications and spatial distribution of subscribers, the preferred choice of packet-switching medium will vary. Intrabuilding applications such as electronic office services may be most economically provided through the use of a coaxial-packet cable system such as the Xerox ETHERNET [18] and LCSNET [64], or twisted pair rings such as DCS [34], coupled with a mix of self-contained user computers (e.g., intelligent terminals with substantial computing and memory capacity) and shared computing, storage, and input-output facilities. Larger area regional applications might best employ shared video cables [35] or packet radios [20], [21] for mobile use. National systems might be composed of a mixture of domestic satellite channels and conventional leased-line services. International systems might use point-to-point links plus a shared communication satellite channel and multiple ground stations to achieve the most cost-effective service.

A consequence of the wide range of technologies which are optimum for different packet-switching applications is that many different networks, both private and public, may co-exist. A network interconnection strategy, if properly designed, will permit local networks to be optimized without sacrificing the possibility of providing effective internetwork services. The potential economic and functional advantages of local networks such as ETHERNET or DCS will lead naturally to private user networks. Such private network developments are analogous to telephone network private automated branch exchanges (PABX) and represent a natural consequence of the marriage of computer and telecommunication technology.

Two further developments can be expected. First, organizations which are dispersed geographically, nationally, or internationally, will want to interconnect these private networks both to share centralized resources and to effect intraorganization electronic mail and other automated office services. Second, there will be an increasing interest in interorganization interconnections to allow automated procurement and financial transaction services, for example, to be applied to interorganization affairs.

In most countries where private networks are permitted, interorganization telecommunication requires the involvement of a PTT. Hence the most typical network interconnection scenarios will involve three or four networks. Within one national administration the private nets of different organizations will be interconnected through a public network. International interconnections will involve at least two public networks. We will return to this topic in Section VI.

In addition to permitting locally optimized networks to be interconnected, a network interconnection strategy should also support the gradual introduction of new networking technology into existing systems without requiring simultaneous global change throughout. This consideration leads to the conclusion that the public data networks should support the most important user requirements for internet service from the outset. If this were the case, then changes in network technology which require a multinetwork system during phased transition would not, *a priori*, have to affect user services.

IV. Provision of End-User Multinetwork Services

The ultimate choice of a network interconnection strategy will be strongly affected by the types of user services which must be supported. It is useful to consider the range of existing and foreseeable user service requirements without regard for the precise means by which these requirements are to be met. We will leave for discussion in subsequent sections the choice of supporting the various services within or external to the packet-switched network. The types of service discussed below are general requirements for network facilities. For this reason they also should be supported across interconnected networks.

Most of the currently prevalent computer-communication services fall into four categories:

1) terminal access to time-shared host computers;
2) remote job entry services (RJE);
3) bulk data transfer;
4) transaction processing.

The time-sharing and transaction services typically demand short network and host response times but modest bandwidth. The RJE and file transfer services more often require high amounts of data transfer, but can tolerate longer delay. Some networks were designed to support primarily terminal service, leaving RJE or file transfer services to be supported by dedicated leased lines. Packet-switching techniques permit both types of service to be supported with common network resources, leading to verifiable economies. However, bulk data transfer requires increasingly higher throughput rates if delivery delays are to be kept constant as the amount of data to be transferred increases.

As distributed operating systems become more prevalent, there will be an increased need for host-to-host transaction services. A prototypical example of such a system is found in the DARPA National Software Works [4], [36]. In such a system, small quantities of control information must be exchanged quickly to coordinate the activity of the distributed components. Broadcast or multidestination services will be needed to support distributed file systems in which information can be stored redundantly to improve the reliability of access and to protect against catastrophic failures.

Transaction services are also finding application in reservation systems, credit verification, point of sale, and electronic funds-transfer systems in which hundreds or thousands of terminals supply to, or request of, hosts small amounts of information at random intervals. Real-time data collection for

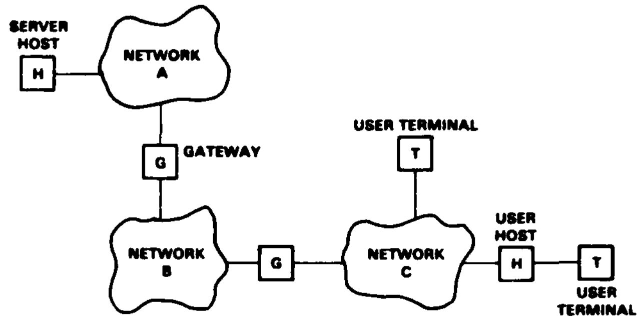


Fig. 1. Network concatenation.

weather analysis, ground and air traffic control, and meter reading, for example, also fall into this category.

More elaborate user requirements can be foreseen as electronic mail facilities propagate. Multiple destination addressing and end-to-end encryption for the protection of privacy as well as support for text, digitized voice, and facsimile message transmission are all likely requirements. Electronic teleconferencing using mixtures of compressed digital packet speech, videographics, real-time cursors (for pointing at video images under discussion), and text display will give rise to requirements for closed user groups and time-synchronized mixes of transaction-like (e.g., for cursor tracking and packet speech) and reliable circuit-like services (e.g., for display management).

Reliability and rapid response will be increasingly important as more and more computer-based applications requiring telecommunications are integrated into the business, government, military, and social fabric of the world economy. The more such systems are incorporated into their daily activities, the more vulnerable the subscribers are to failures. Reliability concerns lead to the requirement for redundant alternatives such as distributed file systems, richly connected networks, and substantial local processing and storage capability. These trends increase the need for networking to share common hardware and software resources (and thus reduce their marginal cost), to support remote software maintenance and debugging, and to support intra- and inter-organizational information exchange.

We have described the end-user services required across one or more data networks. We have carefully refrained from discussing which services should be provided in the data network, and which should be provided in the hosts. Here the choice in single networks will depend on the network technology and the application requirements. For example, in a network using a broadcast technology such as ETHERNET or the SATNET, multidestination facilities may well be incorporated in the data network itself. In typical store-and-forward networks, this feature might be provided at the host level by the transmission of multiple copies of packets. This example highlights immediately the difficulty of using sophisticated services at the data network level across concatenated networks. If *A*, *B*, and *C* are data networks connected as in Fig. 1, and *A* and *C* but not *B* support broadcast or real-time features, it is very difficult to provide them across the concatenation of *A*, *B*, and *C*.

The problem of achieving a useful set of internetwork services might be approached in several ways, as follows.

1) Require all networks to implement the entire range of desired services (e.g., datagram, virtual circuit, broadcast, real-

time, etc.), and then attempt to support these services across the gateways between the networks.

2) Require all networks to implement only the most basic services (e.g., datagram or virtual circuit), support these services across gateways, and rely on the subscriber to implement all other services end-to-end.

3) Allow the subscriber to identify the services which he desires and provide error indications if the networks involved, or the gateways between them, cannot provide the desired services.

4) Allow the subscriber to specify the internetwork route to be followed and depend on the subscriber to decide which concatenation of services are appropriate and what end-to-end protocols are needed to achieve the ultimately preferred class of service.

5) Provide one set of services for local use within each network and another, possibly different set for internetwork use.

The five choices above are by no means exhaustive, and, in fact, only scratch the surface of possibilities. Nothing has been said, thus far, about the compatibility of various levels of communication protocols which exist within each network, within subscriber equipments, and within the logical gateway between networks. To explore these issues further, it will be helpful to have a model of internetwork architecture, taking into account the common principle of protocol layering and the various possible choices of interconnection strategy which depend upon the protocol layer at which the networks are interfaced. We consider this in the next section.

V. Layered Protocol Concepts

Both to provide services in single networks, and to compare the capabilities of different networks, a very useful concept in networking is protocol layering. Various services of increasing capability can be built one on top of the other, each using the facilities of the service layer below and supporting the facilities of the layer above. A thorough tutorial on this concept can be found in the paper by Pouzin and Zimmermann in this issue [37]. We give some specific examples below of layering as a means of illustrating the scope of services and interfaces to be found in packet networks today—and some of the problems encountered in offering services across multiple networks.

Table I offers a very generic view of a typical protocol hierarchy in a store-and-forward computer network, including layers usually found outside of the communication network itself. There are several complications to the use of generic protocol layering to study network interconnection issues. Chief among these is that networks do not all contain the same elements of the generic hierarchy. A second complication is that some networks implement service functions at different protocol layers. For instance, virtual circuit networks implement an end/end subscriber virtual circuit in their intranet, end/end level protocol. Finally, the hierarchical ordering of functions is not always the same in all networks. For instance, TYMNET places a terminal handling protocol within the network access layer, so that hosts look to each other like one or more terminals. Figs. 2-7 illustrate the functional layering of some different networks. It is important to note how the functions vary with the choice of transmission medium.

TABLE I
Generic Protocol Layers

PROTOCOL LAYER	FUNCTIONS
6. APPLICATION	FUNDS TRANSFER, INFORMATION RETRIEVAL, ELECTRONIC MAIL, TEXT EDITING . . .
5. UTILITY	FILE TRANSFER, VIRTUAL TERMINAL SUPPORT
4. END/END SUBSCRIBER	INTERPROCESS COMMUNICATION (E.G. VIRTUAL CIRCUIT, DATAGRAM, REAL-TIME, BROADCAST)
3. NETWORK ACCESS	NETWORK ACCESS SERVICES (E.G. VIRTUAL CIRCUIT, DATAGRAM . . .)
2. INTRANET, END-TO-END	FLOW CONTROL, SEQUENCING
1. INTRANET, NODE-TO-NODE	CONGESTION CONTROL, ROUTING
0. LINK CONTROL	ERROR HANDLING, LINK FLOW CONTROL

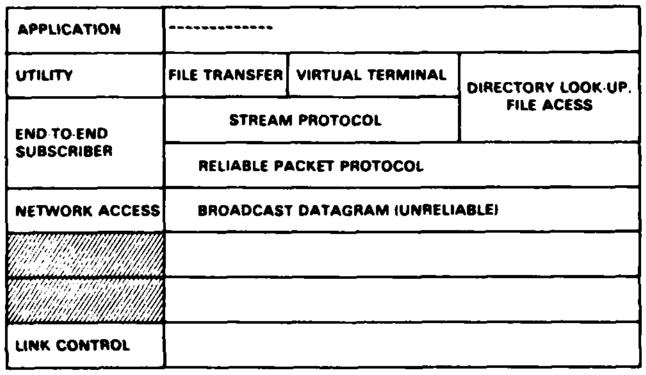


Fig. 2. ETHERNET protocol layering.

A. ETHERNET

In Fig. 2, we represent the Xerox ETHERNET protocol hierarchy. The basic link control mechanism is the ability of the interface device to detect conflict on a shared coaxial cable. If a transmitting interface detects that another interface is also transmitting, it immediately aborts the transmission. Hosts attached to the network interface present datagrams to be transmitted and are told if the datagram was aborted. Datagrams can be addressed to specific interfaces or to all of them. The end/end subscriber layer of protocol is split into two parts: a reliable datagram protocol in which each datagram is reliably delivered and separately acknowledged, and a stream protocol which can be thought of as a virtual circuit. This split is possible, in part, because there is a fairly large maximum datagram size (about 500 bytes) so that user applications can send datagrams without having to fragment and reassemble them. This makes the datagram service useful for many applications which might otherwise have to use the stream protocol. All higher level protocols, such as Virtual Terminal and File Transfer, are carried out in the hosts.

B. ARPANET

The ARPANET protocol hierarchy is shown in Fig. 3. The basic link control between packet switches treats the physical link as eight independent virtual links. This increases effective throughput, but does not necessarily preserve the order in which packets were originally introduced into the network. The intranet node-to-node protocols deal with adaptive routing decisions, store-and-forward service, and congestion control. Hosts have the option of either passing messages (up to

APPLICATION	RJE	ELECTRONIC MAIL	
UTILITY	TELNET	FTP	
END/END SUBSCRIBER	NCP	TCP	NVP/NVCP
NETWORK ACCESS	PERMANENT VIRTUAL CIRCUIT	DATAGRAM	
INTRANET, END/END	FLOW CONTROL, SEQUENCING, MESSAGE REASSEMBLY	▨	
INTRANET, NODE/NODE	ADAPTIVE ROUTING, STORE AND FORWARD, CONGESTION CONTROL		
LINK CONTROL	NON-SEQUENCED, MULTI-CHANNEL ERROR CONTROL		

Fig. 3. ARPANET protocol layering.

8063 bits of text) across the host/network interface, which will be delivered in sequence to the destination, or passing datagrams (up to 1008 bits of text) which are not necessarily delivered in sequence. The user's network access interface is datagram-like in the sense that no circuit setup exchange is needed even to activate the sequenced message service. In effect, this service acts like a permanent virtual circuit over which a sequence of discrete messages are sent. For the sequenced messages, there is exactly one virtual circuit maintained for each host/host pair. In fact, these virtual circuits are set up dynamically and terminated by the source/destination packet switches so as to improve resource utilization [38], [62].

The end/end subscriber layer of ARPANET contains two main protocols: Network Control Protocol (NCP, [39], [40]) and Transmission Control Protocol (TCP, [25]). NCP was the first interprocess communication protocol built for ARPANET. It relies on the sequenced message service provided by the network and derives multiple virtual circuits between pairs of hosts by multiplexing. The TCP can use either the sequenced message service or the datagram service. It does its own sequencing and end/end error control and derives multiple virtual circuits through extended addressing and multiplexing. TCP was designed for operation in a multinet environment in which the only service which reasonably could be expected was an unreliable, unsequenced datagram service.

To support experiments in packetized voice communication, two protocols were developed for use on the ARPANET. The Network Voice Protocol (NVP) and Network Voice Conferencing Protocol (NVCP) use the datagram service to achieve very low delay and interarrival time variance in support of digital, compressed packet speech (more on these protocols may be found in [41]). The NVP could be considered the basis for a generic protocol which could support a variety of real-time, end/end user applications.

The higher level utility protocols such as terminal/host protocol (TELNET, [40], [42]) and file transfer protocol (FTP, [40], [42]) use virtual circuits provided by NCP or TCP. The FTP requires one live interactive stream to control the data transfer, and a second for the data stream itself. Yet higher level applications such as electronic mail and remote job entry (RJE, [40], [42]) use mixtures of TELNET and FTP to effect the service desired. These protocols are usually put into the hosts. There is one anomaly, which occurs in many networks. Because terminal handling is required so frequently, a Terminal Interface Message Processor (TIP, [43]) was built. This device is physically integrated with the packet switch (IMP, [38]); it includes also the NCP and TELNET protocols.

END/END SUBSCRIBER	TERMINAL-TO-HOST
NETWORK ACCESS	VIRTUAL CIRCUIT
INTRANET END-END	▨
INTRANET NODE-NODE	FRAME DISASSEMBLY, REASSEMBLY, ROUTING, STORE/FORWARD, CONGESTION CONTROL
LINK CONTROL	FRAME BASED ERROR CONTROL, RETRANSMISSION, SEQUENCING

Fig. 4. TYMNET protocol layering.

C. TYMNET

TYMNET (see Fig. 4) is one of the oldest of the networks in the collection described here [3]. Strictly speaking, it operates rather differently than other packet-switched networks, because the frames of data that move from switch to switch are disassembled and reassembled in each switch as an integral part of the store-and-forward operation. Nevertheless, the network benefits from the asynchronous sharing of the circuits between the switches in much the same way that more typical packet-switched networks do. The network was designed to support remote terminal access to time-shared computer resources. The basic service is the transmission of a stream of characters between the terminal and the serving host. A frame is made up of one or more blocks of characters, each block labeled with its source terminal identifier and length. The switch-to-switch layer of protocol disassembles each frame into its constituent blocks and uses a routing table to determine to which next switch the block should be sent. Blocks destined for the same next switch are batched together in a frame which is checksummed and sent via the link control procedure to the next switch. Batching the blocks reduces line overhead (the blocks share the frame checksum) at the expense of more CPU cycles in the switch for frame disassembly and reassembly.

The protocol between TYMNET switches also includes a flow control mechanism which, because of the fixed routes, can be used to apply back pressure all the way back to the traffic source. This is not precisely an end-to-end flow control mechanism, but a hop-by-hop back pressure strategy. Character blocks are kept in sequence along the fixed routes so that no resequencing is required as they exit from the network at their destinations. The network interface is basically a virtual circuit designed to transport character streams between a host and a terminal. The same virtual circuits can be used to transport character streams between hosts, which look to each other like a collection of terminals. Above the basic virtual circuit service, is a special echo-handling protocol which allows the host and the terminal handler in the "remote TYMSAT" to coordinate the echoing of the characters typed by a user.

D. PTT Networks

Many PTT networks, e.g., TELNET, TRANSPAC, DATAPAC, and EURONET use a particular network-access protocol, X.25 [28], [29] (see Fig. 5). This protocol has been recommended by the CCITT for public packet-switched data networks. X.25 is a three-part protocol consisting of a hardware electrical interface, X.21 [44], the digital equivalent of the usual V.24 or EIA-RS232C modem interface [45], a link control procedure, High Level Data Link Control (HDLC, [46]), and a packet-level protocol for effecting the setup, use, termination, flow, and error control of virtual circuits.

UTILITY	TERMINAL HANDLING X.28, X.29
END/END SUBSCRIBER	▨
NETWORK ACCESS	X.25. PERMANENT OR TEMPORARY VIRTUAL CIRCUITS
INTRANET, END END	MULTIPLE VIRTUAL CIRCUITS, FLOW CONTROL
INTRANET NODE NODE	ROUTING STORE/FORWARD, CONGESTION CONTROL
LINK CONTROL	HDLC OR EQUIVALENT

Fig. 5. PTT protocol layering.

In all but the DATAPAC network, a fixed route for routing packets through the network is selected at the time the virtual circuit is created. "Permanent" virtual circuits are a customer option; if used, the setup phase is invoked only in the case of a network failure. Between source and destination packet switches, a virtual circuit protocol is operated which implements end-to-end flow control on multiple virtual circuits between pairs of packet switches. Up to 4096 virtual circuits between pairs of host ports can be maintained by each packet switch, as compared to the single virtual circuit provided by ARPANET (on which hosts can multiplex their own virtual circuits). This choice has a noticeable impact on the subscriber interface protocol which becomes complicated because the subscriber host and the packet switch to which it attaches must maintain a consistent view of the state of each virtual circuit in use.

To provide for echo control, user commands, code conversion, and other terminal-related services, these networks implement CCITT Recommendations X.28 [29] and X.29 [29] in a PAD (Packet Assembly and Disassembly unit). These protocols sit atop the virtual circuit X.25 protocol. In order to serve customers desiring a terminal-to-host service with character terminals, such as is provided by TYMNET or by the ARPANET (through the TIP), most of the PTT networks mentioned are developing a PAD unit. A matching X.29 (PAD control protocol) layer must be provided in hosts offering to service terminals connected to PAD's.

E. High Level Protocols

The X.25/X.28/X.29 protocol hierarchy does not include an end/end subscriber or high-level protocol layer. Some customers will, in fact, implement end-to-end protocols on top of the virtual circuit protocol, but others may not. Several attempts are being made to standardize protocols above the network access level. The ARPANET community has developed a Transmission Control Protocol [25] for internetwork operation to replace the Network Control Program (NCP) developed early in the ARPANET project. The International Federation of Information Processing (IFIP) has proposed a Transport Station through its Working Group 6.1 on Network Interconnection [47]; the proposal has been submitted to the International Standards Organisation (ISO) as a draft standard. In addition, other communities, e.g., the High Level Protocol Working Group in the UK, have devised protocols for Virtual Packet Terminals (VPT, [48]) and File Transport Protocol (FTP, [49]) which are intended to be network independent and which may be submitted to CCITT. The ISO study on "open systems architecture" and the proposed similar study by CCITT Study Group VII will attempt to evolve higher level protocol recommendations for existing and future data networks.

This brief summary of different network-protocol layerings is in no way comprehensive, but illustrates the diversity of protocol designs which can be found on nets providing different types of services to subscribers.

VI. Technical Interconnection Choices

A. The Issues

Beginning with the earliest papers dealing with strategies for packet-network interconnection [23]-[26], [32], the common objective of all the proposed methods is to provide the physical means to access the services of a host on one network to all subscribers (including hosts) of all the interconnected networks. Of course, limitations to this accessibility are envisaged, imposed either for administrative reasons or by the scarcity of resources. The achievement of this objective invariably requires that data produced at a source in one net be delivered and correctly interpreted at the destination(s) in another network. In an abstract sense, this boils down to providing interprocess communication across network boundaries. Even if a person is the ultimate source of the data, packet-switching networks must interpose some degree of software processing between the person and the destination service, even if only to assemble or disassemble packets produced by a computer terminal.

A fundamental aspect of interprocess communication is that no communication can take place without some agreed conventions. The communicating processes must share some physical transmission medium (wire, shared memory, radio spectrum, etc.), and they must use common conventions or agreed upon translation methods in order to successfully exchange and interpret the data they wish to communicate. One of the key elements in any network interconnection strategy is therefore how the required commonality is to be obtained. In some cases, it is enough to translate one protocol into another. In others, protocols can be held in common among the communicating parties.

In any real network interconnection, of course, a number of secondary objectives will affect the choice of interconnection strategy. For example achievable bandwidth, reliability, robustness (i.e., resistance to failures), security, flexibility, accountability, access control, resource allocation options, and the like can separately and jointly influence the choice of interconnection strategy. Combinations of strategies employing protocol standards and protocol translations at various levels of the layered protocol hierarchy are also likely possibilities.

There are a number of issues which must be resolved before a coherent network interconnection strategy can be defined. A list of some of these issues, which will be treated in more detail in succeeding sections, is:

1) level of interconnection;
2) naming, addressing, and routing;
3) flow and congestion control;
4) accounting;
5) access control;
6) internet services.

B. Gateways and Levels of Network Interconnection

The concept of a gateway is common to all network interconnection strategies. The fundamental role of the gateway is to terminate the internal protocols of each network to which it is attached while, at the same time, providing a common

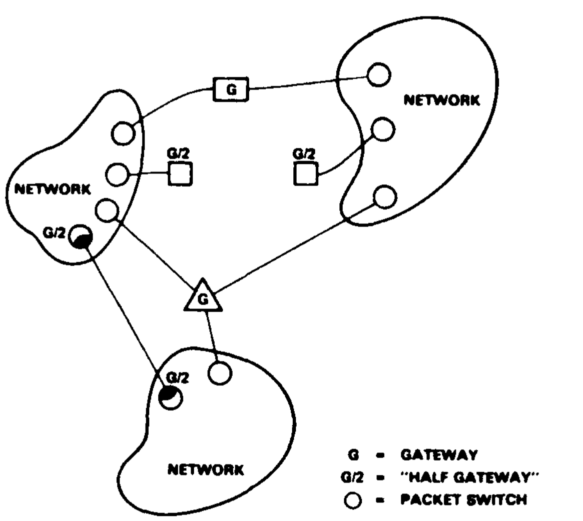


Fig. 6. Various gateway configurations.

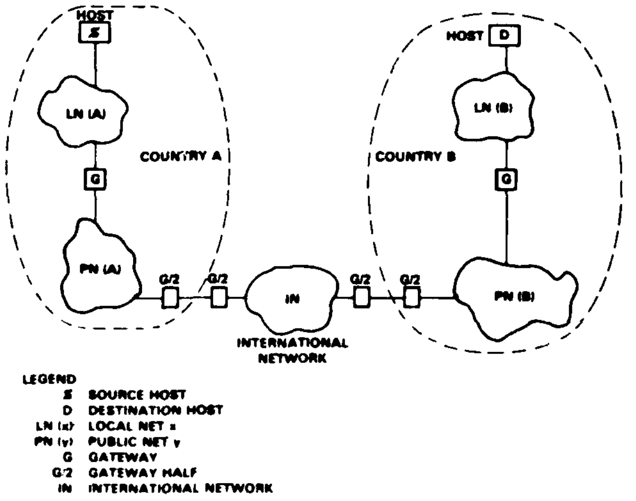


Fig. 7. International packet-networking model.

ground across which data from one network can pass into another. However, the choice of functions to be performed in the gateway varies considerably among different interconnection strategies (see Fig. 6). The term "gateway" need not imply a monolithic device which joins a pair of networks. Indeed, the gateway may merely be software in a pair of packet switches in different networks, or it may be made up of two parts, one in each network (a sort of "gateway half"). In the latter case, the two halves might be devices separate and distinct from the network packet switches or might be integrated with them. Furthermore, a gateway might interconnect more than two networks. In the material which follows, every attempt has been made to avoid any implicit choice of gateway implementation. It is worth pointing out, however, that the "half gateway" concept is highly attractive from both a technical and a purely administrative point of view. Technically, each half could terminate certain levels of protocol of the net to which it is attached. Administratively each half could be the responsibility of the network to which it belongs. Then the only matters for jurisdictional negotiation are the physical medium by which the half-gateways exchange data, and the format and protocol of the exchange.

It is important to realize that typical applications may involve three or more networks. Where local networks are used, they will usually need to be interconnected to realize the benefits of interorganizational data exchange. In most countries, such interconnections will only be permitted through a public network. Thus for a typical national situation, three networks and two gateways will be involved in providing the desired host-to-host communication.

The international picture is similar, except that more networks are likely to be involved. Shown in Fig. 7, the path from a host, S, on local network $LN(A)$ in country A, passes through a public network, $PN(A)$ in country A, through an international network IN, through a public network $PN(B)$ in country B, and finally through a local network, $LN(B)$, to the destination host, D. There are four internetwork gateways involved. It is this model involving multiple gateways that guides us away from network interconnection methods which rely on the source and destination hosts being in adjacent networks connected by the mediation of a single gateway.

1) Common Subnet Technology (Packet Level Interconnection): The level at which networks are interconnected can be determined by the protocol layers terminated by the gateway. For example, if a pair of identical networks were to be interconnected at the interpacket-switch level of protocol, we might illustrate the gateway placement as shown in Fig. 8. Here the "gateway" may consist only of software routines in the adjacent packet switches, e.g., $P(A)$ and $P(B)$, which provide accounting, and possibly readdressing functions. The contour model of protocol layer is useful here since it shows which levels are common to the two networks and which levels could be different. In essence, those layers which are terminated by the gateways could be different in each net, while those which are passed transparently through the gateway are assumed to be common in both networks. This network interconnection strategy requires that the internal address structure of all the interconnected networks be common. If, for example, addresses were composed of a network identifier, concatenated with a packet-switch identifier and a host identifier, then addressing of objects in each of the networks would be straightforward and routing could be performed on a regional basis with the network identifiers acting as the regional identifiers, if desired. Alternatively, two identical networks could adopt a common network name and assign nonduplicative addresses to each of the packet switches in both networks. This may require that addresses in one network be changed.

The strategy described above might be called the "common subnetwork strategy," since, in the end, subscribers of the newly formed joint network would essentially see a single network. This strategy does not rule out the provision of special access control mechanisms in the gateway nodes which could filter traffic flowing from one network into the other. Similarly, the gateway nodes could perform special internetwork traffic accounting which might not normally be performed in a subnet switching node. This network interconnection method is limited to those cases in which the nets to be connected are virtually identical, since the gateways must participate directly in all the subnet protocols. The end-to-end subnet protocols (e.g. source/destination packet-switch protocols) must pass transparently through the gateways to permit interactions between a source packet switch in one net and a destination packet switch in another. The resulting network presents the same network access interface to all

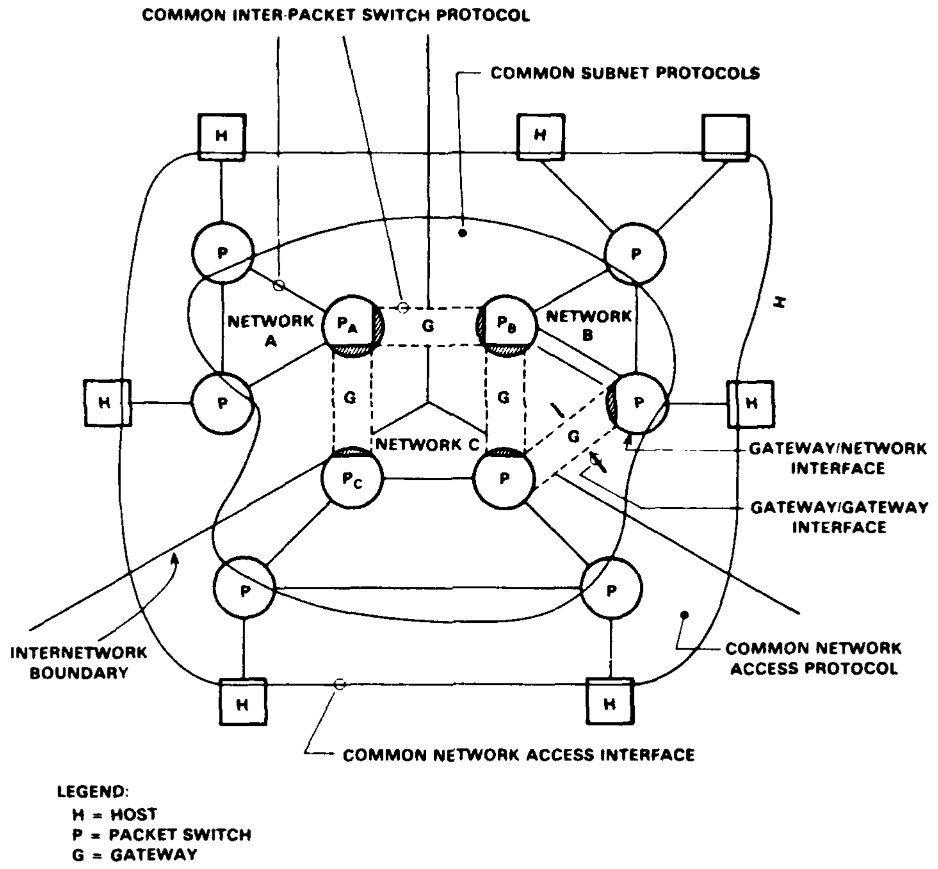

COMMON INTER-PACKET SWITCH PROTOCOL
COMMON SUBNET PROTOCOLS
H
P
NETWORK A
P_A
G
P_B
NETWORK B
NETWORK C
P_C
GATEWAY/NETWORK INTERFACE
GATEWAY/GATEWAY INTERFACE
INTERNETWORK BOUNDARY
COMMON NETWORK ACCESS PROTOCOL
COMMON NETWORK ACCESS INTERFACE
LEGEND:
H = HOST
P = PACKET SWITCH
G = GATEWAY

Fig. 8. Interconnection of common subnetworks.

subscribers, and this leads us to the next example which is based on the concept of a common network access interface.

2) Common Network Access Interfaces: If the subnetwork protocols are not identical, the next opportunity to establish internetwork commonality is at the network access interface. This is illustrated in Fig. 9. Each network is assumed to have its own intranet protocols. However, each network presents the same external interface to subscribers. This is illustrated by showing a common interface passing through all hosts, marked "common network access interface" in the figure.

Once again, the gateway could be thought of as software in adjacent packet switches. Each gateway is composed of two halves formed by linking the packet switches of two nets together. However, in this case, the subnetwork protocols are terminated at the gateway so that the intergateway exchange looks more like network access interaction than a node-to-node exchange. This is the approach taken by CCITT with its X.25 packet network interface recommendation and X.75 intergateway exchange recommendation.

It is important to note that the intergateway interface could be similar to the standard network access interface, but it need not necessarily be identical.

There are two basic types of network interface currently in use: 1) the datagram interface [31]; and 2) the virtual circuit interface [32]. The details of these generic interface types vary in different networks; some networks even offer both types of interface. In some, the interface to use may be chosen at subscription time; in others it may be possible for a subscriber to select the access method dynamically.

A datagram interface allows the subscriber to enter packets into the network independent of any other packets which have been or will be entered. Each packet is handled separately by the network. A virtual circuit interface requires an exchange of control information between the subscriber and the network for the purpose, for example, of setting up address translation tables, setting up routes or preallocating resources, before any data packets are carried to the destination. Some networks may implement a *fast select* virtual circuit interface in which a circuit setup request is sent together with the first (and possibly last) data packet. Other control exchanges would be used to close the resulting virtual circuits set up in this fashion.

It is essential to distinguish datagram and virtual circuit services from datagram and virtual circuit interfaces. A datagram service is one in which each packet is accepted and treated by the network independently of all others. Sequenced delivery is not guaranteed. Indeed, it may not be guaranteed that all datagrams will be delivered. Packets may be routed independently over alternate network paths. Duplicate copies of datagrams might be delivered.

Virtual circuit service tries to guarantee the sequenced delivery of the packets associated with the same virtual circuit. It typically provides to the host advice from the network on flow control per virtual circuit as opposed to the packet-by-packet acceptance or rejection typical of a datagram service. If the network operation might produce duplicate packets, these are filtered by the destination packet switch before delivery to the subscriber. Duplicate packet creation is a

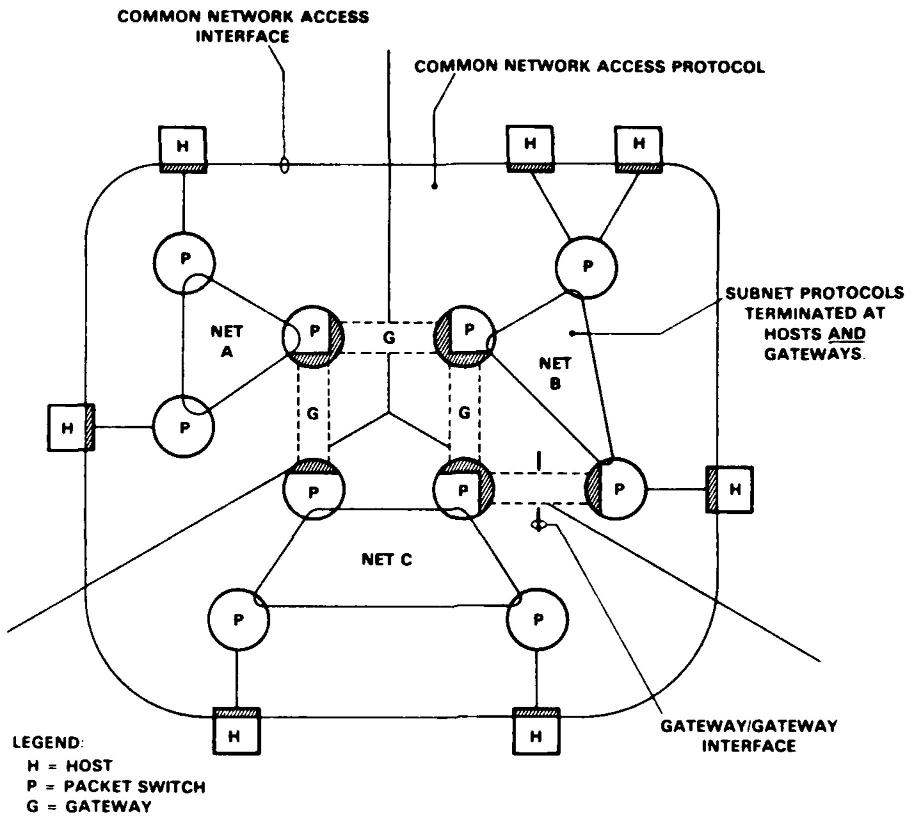

COMMON NETWORK ACCESS INTERFACE
COMMON NETWORK ACCESS PROTOCOL
H
P
NET A
G
NET B
NET C
SUBNET PROTOCOLS TERMINATED AT HOSTS AND GATEWAYS.
GATEWAY/GATEWAY INTERFACE
LEGEND:
H = HOST
P = PACKET SWITCH
G = GATEWAY

Fig. 9. Interconnection of networks with common network-access interfaces.

common phenomenon as in packet-switched store-and-forward systems. The basic mode of operation is to forward a packet to the next switch and await an acknowledgment. After a timeout, the packet is retransmitted. If an acknowledgment is lost due to line noise, for example, then two copies of the packet would have been transmitted. Even if the next switch is prepared to filter duplicates out, a network which uses adaptive routing can deliver a duplicate packet to the periphery of the network. For example, if a packet switch receives a packet successfully but the line to the sender breaks before the receiver can acknowledge, the sender may send another copy to a *different* packet switch. Both packet copies may be routed and delivered to the destination packet switch where final duplicate filtering would be needed if virtual circuit service is being provided.

Some networks offer both a datagram and a virtual circuit service; some offer a single interface, but different services. For example, the ARPANET has a basic datagram interface. However, the subnetwork will automatically provide a sequenced virtual circuit service (i.e., packets are kept in sequence when they are delivered to the destination) if the packet is marked appropriately. Otherwise, packets are not delivered in sequence nor are packet duplicates or losses, except for line by link correction, recovered within the network for nonsequenced types of traffic.

By contrast, TRANSPAC offers a virtual circuit interface and service. Subscribers transmit "call request" packets containing the full destination address to the packet switch. The request packet is forwarded to the destination, leaving behind a fixed route. The destination subscriber returns a "call accepted" packet which is delivered to the caller. As a result of this exchange, the source subscriber has associated a "logical channel number" or LCN, with the full source-destination addresses. Thus subsequent packets to be sent on the same logical channel are identified by the LCN and are kept in sequence when delivered to the destination.

Finally, it is possible to implement a datagram-like service using a virtual circuit interface. In this case, the exchange of *request* and *accept* packets might be terminated at the subscriber's local packet switch, so that even if packets were not delivered in sequence they might employ abbreviated addressing for local subscriber and packet-switch interaction.

If network interaction is to be based on a standard interface, then agreement must be reached both on the interface and an associated service or services. Furthermore, a common addressing system is needed so that a subscriber on one network can address a packet to a subscriber on any other network. A weaker assumption could be made but we are deliberately assuming a truly common service, interface, and addressing mechanism. We will return to this topic in a later section.

The choice of a standard network service through which to effect network interconnection has a primary impact on the flexibility of implementable network interconnection methods. We will consider two choices: datagr.m service and virtual circuit service.

a) Datagram service as a standard for network interconnection: For this case, it is assumed that every network offers a common datagram service. A uniform address space makes it possible for subscribers on any network to send packets addressed to any ·her subscriber on a connected network. Packets are routed between subscriber and gateway and between gateways based on the destination address. No attempt is made to keep the datagrams in any order in transit or upon delivery to the destination. Individual datagrams may be freely routed through different gateways to recover from failures or to allow load-splitting among parallel gateways joining a pair of networks.

The gateway/gateway interface may be different than the network access interface, if need be (see Fig. 9).

This strategy requires that all networks implement a common interface for subscribers. The simplicity and flexibility of the datagram interface strategy is offset somewhat by the need for all networks to implement the same interface. This is true for the pure virtual circuit interface strategy as well, as will be shown below.

One of the problems which has to be faced with any network interconnection strategy is congestion control at the gateways. If a gateway finds that it is unable to forward a datagram into the next network, it must have a way of rejecting it and quenching the flow of traffic entering the gateway en route into the next network. The quenching would typically take the form of an error or flow control signal passing from one gateway half to another on behalf of the associated network. Similar signals could be passed between subscribers and the packet network for similar reasons. Since datagram service does not undertake to guarantee end/end reliability, it is possible to relieve momentary congestion by discarding datagrams, as a last resort.

b) Virtual circuits for network interconnection: Another alternative standard network service which could be used for network interconnection is virtual circuit service (Fig. 10). Independent of the precise interface used to "set up" the virtual circuit, a number of implementation issues immediately arise if such a service is used as a basis for network interconnection.

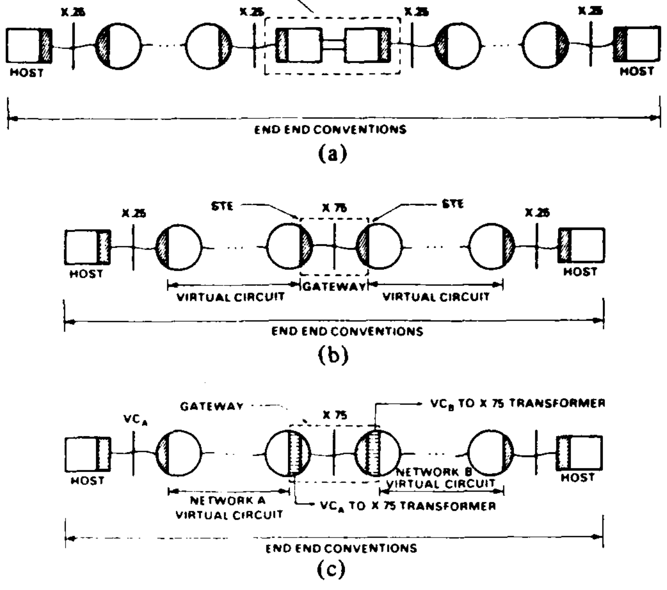


Fig. 10. Virtual circuit network interconnection strategies. (a) Subscriber-based gateway. Internet source and destination carried in user data field of X.25 call set-up packets. (b) X.75 based gateway. Note how much of the X.25 VC service is terminated at the STE. (c) X.75-based gateways with general virtual circuit networks.

Since it is intended that all packets on a virtual circuit be delivered to the destination subscriber in the same sequence as they were entered by the source subscriber, it is necessary that either: 1) all packets belonging to the same virtual circuit take the same path from source subscriber, through one or more gateways, to destination subscriber; or 2) all packets contain sequence numbers which are preserved end-to-end between the source DCE in the originating network and the destination DCE in the terminating network.

In the first case, virtual circuits are set up and anchored to specific gateways so that the sequencing of the virtual circuit service of each network can be used to preserve the packet sequence on delivery. This results in the concatenation of a series of virtual circuits through each gateway and, therefore, the knowledge of each virtual circuit at each gateway (since the next gateway to route the packet through must be fixed for each virtual circuit).

In the second case, there is no need to restrict the choice of gateway routing for each virtual circuit since the destination DCE will have sufficient information to resequence incoming packets prior to delivery to the destination subscriber.

In either case, the destination DCE will have to buffer and resequence packets arriving out of order due either to disordering within the last network or to alternate routing among networks, if this is permitted. Some networks may keep packets in sequence as they transit the network. This will only be advantageous at the destination DCE if the packets enter the network in the desired sequence. If such a service is relied upon in the internet environment, then each gateway must assure that on entry to such a net, the packets are in the desired order for delivery to a destination subscriber or another gateway.

The buffering and resequencing of packets within the networks or at gateways introduces substantial variation in buffer space requirements, packet transit delays, and the potential for buffer lockups to occur [50], [51], [61].

If packets for a specific virtual circuit are restricted to pass through a fixed series of gateways, and if a standard flow-control method is agreed upon as part of the virtual circuit service, then it is possible for each internet gateway to participate in end-to-end flow control by modifying the flow control information carried in packets carried end-to-end from the source DCE to the destination DCE. Consequently, a gateway may be able to adjust the amount of traffic passing through it and thereby achieve a kind of internet gateway congestion control. If this is done by allocating buffer space for "outstanding" packets, then either the gateways must guarantee the advertised buffer space or there must be a retransmission capability built into the internet virtual circuit implementation, perhaps between source DCE and destination DCE or between DCE's and gateways.

Such a mechanism does not, however, solve the problem of network congestion unless the gateway-flow control decisions take into account resources both in the gateway and in the rest of the network. Although it is tempting to assume that virtual circuit-flow control can achieve internetwork congestion control, this is by no means clear, and is still the subject of considerable research.

As a general rule, compared to the datagram method, the virtual circuit approach requires more state information in each gateway, since knowledge of each virtual circuit must be maintained along with flow control and routing information. The usual virtual circuit interface is somewhat more complex for subscribers to implement as well, because of the amount of state information which must be shared by the subscriber and the local DCE. For example, implementations of the X.25

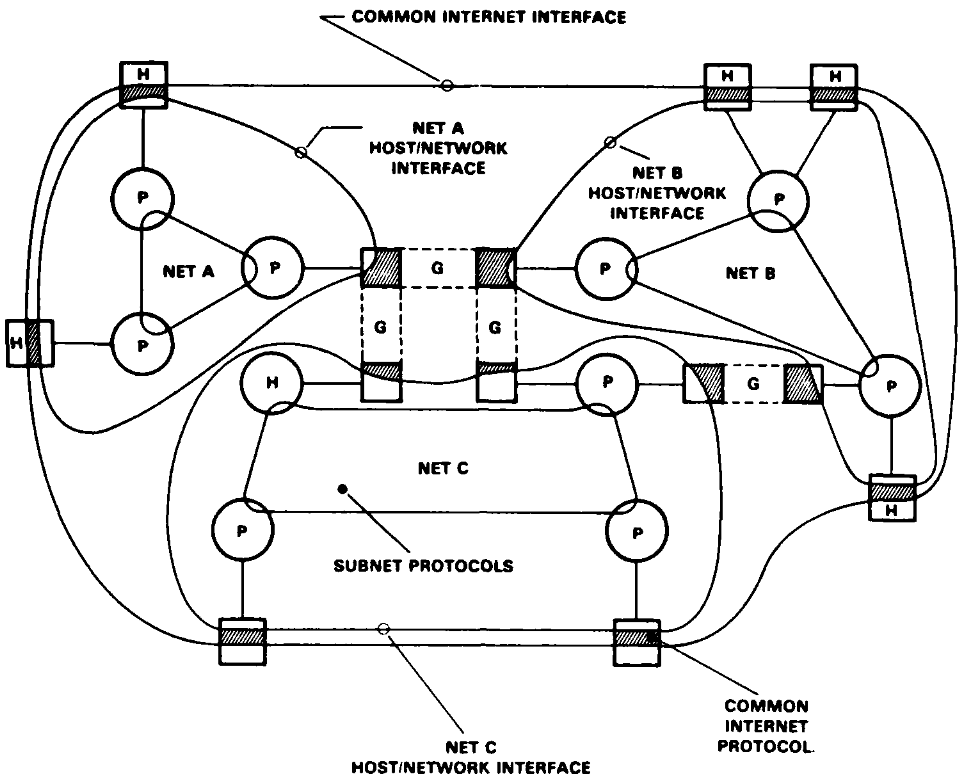

COMMON INTERNET INTERFACE
NET A HOST/NETWORK INTERFACE
NET B HOST/NETWORK INTERFACE
NET A
NET B
NET C
SUBNET PROTOCOLS
H
P
G
COMMON INTERNET PROTOCOL
NET C HOST/NETWORK INTERFACE

Fig. 11. Common internet interface.

interface protocol have been privately reported by Computer Corporation of America and University College London to require 4000–8000 words of memory on Digital Equipment Corporation PDP-11 computers. By contrast, the ARPANET and Packet Radio Network datagram interfaces require 500–1000 words of memory on the same machine. For internetwork operation, this may be even more burdensome, since any failure at a gateway may require a subscriber-level recovery through an end-to-end protocol, in addition to the virtual circuit interface software, as is shown in [52].

Nevertheless, it may be advantageous to consider internetworking standards which usefully employ both datagram and virtual circuit interfaces and services. For example, some special internet services such as multidestination delivery may be more efficient if they are first set up by control exchanges between the subscriber and the local network and perhaps gateways as well. Once set up, however, a datagram mode of operation may be far more efficient than maintaining virtual circuits for all destinations. Implicit virtual circuits which are activated by simple datagram-like interfaces are also attractive for very simple kinds of terminal equipment.

If it is not possible for all networks to implement a common network-access interface, then the next opportunity is to standardize only the objects which pass from one net to the next and to minimize any requirements for the sequencing of these objects as they move from net to net.

3) General Host Gateways: In this model, a gateway is indistinguishable from any other network host and will implement whatever host/network interface is required by the networks to which it is attached. For many networks, this may be X.25, but the strategy does not rely on this. The principle assumption is that packet networks are at least capable of carrying subscriber packets up to some maximum length, which may vary from network to network. It is specifically not assumed that these packets will be delivered in order through intermediate networks and gateways to the destination host. This minimal type of service is often termed "datagram" service to distinguish it from sequenced virtual circuit service. A detailed discussion of the tradeoff between datagram and virtual circuit types of networks is given elsewhere [52].

The basic model of network interconnection for the datagram host gateway is that internetwork datagrams will be carried to and from hosts and gateways and between gateways by encapsulation of the datagrams in local network packets. Pouzin describes this process generically as "wrapping" [37]. The basic internetwork service is therefore a datagram service rather than a virtual circuit service. The concept is illustrated in Fig. 11.

Datagram service does not offer the subscriber as many facilities as virtual circuit service. For example, not all datagrams are guaranteed to be delivered, nor do those that are delivered have to be delivered in the sequence they were sent. Virtual circuits, on the other hand, do attempt to deliver all packets entered by the source in sequence to the destination. These relaxations allow dynamic routing of datagrams among multiple, internetwork gateways without the need for subscriber intervention or alert.

The internet datagram concept gives subscribers access to a basic internet datagram service while allowing them to build more elaborate end-to-end protocols on top of it. Fig. 12 illustrates a possible protocol hierarchy which could be based on the internet datagram concept. The basic internet datagram service could be used to support transaction protocols or real-time protocols (RTP) such as packet-voice protocols (PVP) which do not require guaranteed or sequenced data

UTILITY	FTP	VTP	RTP	VP
END/END SUBSCRIBER	END/END VIRTUAL CIRCUIT		END/END DATAGRAM	
INTERNET ACCESS	INTERNET DATAGRAM			
NETWORK ACCESS	NETWORK SPECIFIC			
INTRANET, END-END	NETWORK SPECIFIC			
INTRANET, NODE-NODE	NETWORK SPECIFIC			
LINK CONTROL	NETWORK SPECIFIC			

Fig. 12. Protocol layering with internetwork datagrams.

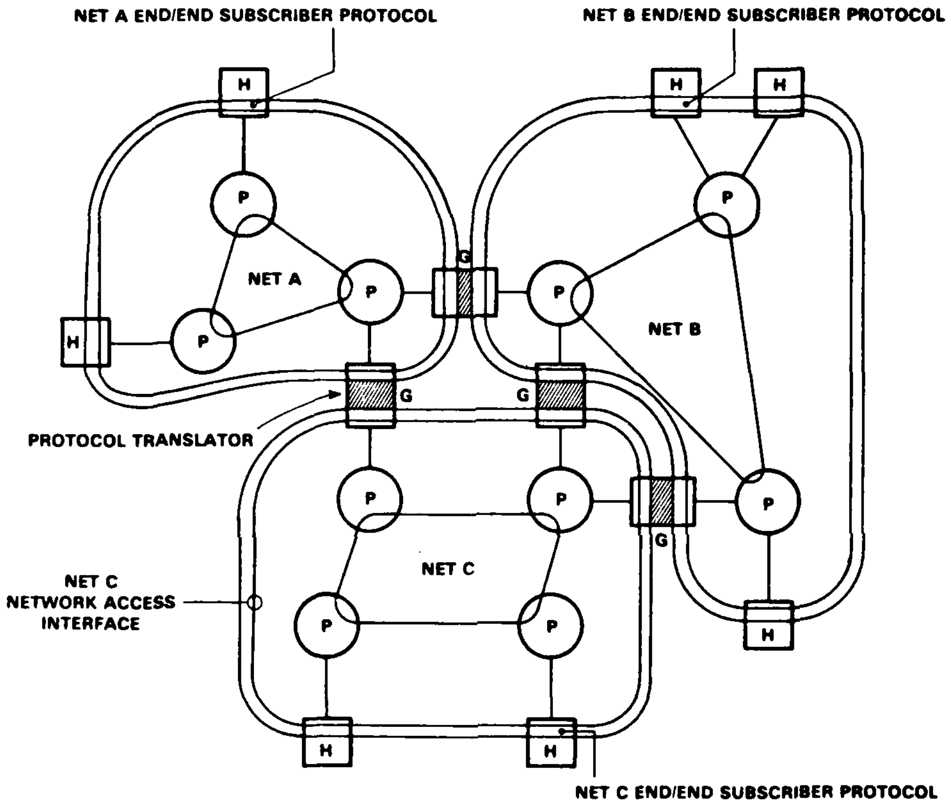

NET A END/END SUBSCRIBER PROTOCOL
NET B END/END SUBSCRIBER PROTOCOL
H
P
NET A
NET B
G
PROTOCOL TRANSLATOR
NET C
NET C NETWORK ACCESS INTERFACE
NET C END/END SUBSCRIBER PROTOCOL

Fig. 13. Host protocol translation gateway.

delivery; reliable, sequenced protocols could be constructed above the basic internet datagram service to perform end/end sequencing and error handling. Applications such as virtual terminal protocols (VTP) [40], [42], [48] or file-transfer protocols [40], [42], [49] could be built above a reliable, point-to-point, end/end service which is itself built atop internet datagrams. Under this strategy, the basic gateway functions are the encapsulation and decapsulation of datagrams, mapping of internet source/destination addresses into local network addresses and datagram routing. Gateways need not have any knowledge of higher level protocols if it is assumed that protocols above the internet datagram layer are held in common by the communicating hosts. Datagrams can be routed freely among gateways and can be delivered out of sequence to the destination host.

The basic advantage of this strategy is that almost any sort of network can participate, whether its internal operation is datagram or virtual circuit oriented. Furthermore, the strategy offers an easy way for new networks to be made "backwards compatible," with older ones while allowing the new ones to employ new internal operations which are innovative or more efficient.

Every subscriber must implement the internet datagram concept for this strategy to work, of course. The same problem arises with the standard network interface strategy since all subscribers must implement the same network interface.

4) Protocol Translation Gateways: It would be misleading to claim that the concept of protocol translation has not played a role in the discussion thus far. In a sense, the encapsulation of internet datagrams in the packet format of each intermediate network is a form of protocol translation. The basic packet carrying service of one network is being translated into the next network's packet carrying service (see Fig. 13). This concept could be extended further. For example, if two networks have a virtual circuit concept, one implemented within the subnetwork and the other through common host/host protocols, it might be possible, at the gateway between the nets, to map one network's virtual circuit into the other's. This same idea could be applied to higher level protocol mappings as well; for instance, the virtual terminal protocol for one network might be transformed into that of another "on the fly."

The success of such a translation strategy depends in large part on the commonality of concept between the protocols to be translated. Mismatches in concept may require that the service obtained in the concatenated case be a subset of the services obtainable from either of the two services being translated. Extending such translations through several gateways can be difficult, particularly if the protocols being translated do not share a common address space for internetwork sources/destinations. In the extreme, this strategy can result in subscribers "logging in" to the gateway in order to activate the protocols of the next network. Indeed, front-end computers could be considered degenerate translation gateways since they transform host/front-end protocols into network protocols.

There are circumstances when translation cannot be avoided. For instance, when the protocols of one network cannot be modified, but internet service is desired, there may be no alternative but to implement protocol translations. The model typically used to guide protocol translation gateways is that the source/destination hosts lie on either side of the translation gateway. Concatenation of protocol translations through several networks and gateways is conceivable, but may be very difficult in practice and may produce very inefficient service.

C. Names, Addresses, and Routes

In order to manage, control, and support communication among computers on one or more networks, it is essential that conventions be established for identifying the communicators. For purposes of this discussion, we will use the term *host* to refer to all computers which attach to a network at the network-access level of protocol (see Table I). Subscribers to terminal-access services can be thought of as attaching to hosts, even if the host is embedded in the hardware and software of a packet switch as a layer of protocol. Consequently, we can say that the basic task of a packet-switching network is to transport data from a source host to one or more destination hosts.

To accomplish this task, each network needs to know to which destination packets are to be delivered. Even in broadcast nets such as the ETHERNET, this information is necessary so that the destination host can discriminate packets destined for itself from all others heard on the net. At the lowest-protocol levels it is typical to associate destinations with *addresses*. An address may be simply an integer or it may have more internal structure.

At higher levels of protocol, however, it is more common to find text strings such as "MULTICS" or "BBN-TENEX" used as *names* of destinations. Application software, such as electronic mail services, might employ such names along with more refined destination identifiers. For example, one of the authors has an electronic mailbox named "KIRSTEIN at ISI" located in a computer at the University of Southern California's Information Sciences Institute.

Typically, application programs transform names into addresses which can be understood by the packet-switching network. The networks must transform these addresses into *routes* to guide the packets to their destination. Some networks bind addresses to routes in a relatively rigid way (e.g., setting up virtual circuits with fixed routing) while others determine routes as the packets move from switch to switch, choosing alternate routes to bypass failed or congested areas of the network. Broadcast networks need not create routes at all (e.g., SATNET).

In simple terms, a *name* tells what an object is; an *address* tells where it is; and a *route* tells how to get there [54]. A simple model involving these three concepts is that hosts transform names into addresses and networks transform addresses into routes (if necessary). However, this basic model does leave a large number of loose ends. The subject is so filled with issues that it is not possible in this paper to explore them all in depth. In what follows, some of the major issues are raised and some partial resolutions are offered.

One major question is "Which objects in the network should have names? addresses?" Pouzin and Zimmermann offer a number of views on this question in their paper in this issue [37]. A generic answer might be that at least all objects which can be addressed by the network should have names as well so that high-level protocols can refer to them. For example, it might be reasonable for every host connection on the network to have an name and an address. There also may be objects internal to the network which also have addresses such as the statistics-gathering *fake hosts* in the ARPANET [38].

A related issue is whether objects should or can have multiple names, multiple addresses, and multiple routes by which they can be reached. The most general resolution of this issue is to permit multiple names, addresses, and routes to exist for the same object. An example taken from the multinetwork environment may serve to illustrate this notion. Fig. 6 shows three networks which are interconnected by a number of gateways. Each gateway (or pair of gateway halves) has two interfaces, one to each network to which it is attached. Plainly there is the possibility that several alternate routes passing through different gateways and networks could be used to carry packets from a source host in one net to a destination host in another net. This is just the analog of alternate routing within a single network.

Furthermore, each gateway has two addresses, typically one for each attached network. This is just the analog of a host on one network attached to two or more packet switches for reliability. The term *multihoming* is often used to refer to multiply attached hosts.

Finally, it may be useful to permit a gateway to have more than one name, for example, one for each network to which it is attached. This might allow high-level protocols to force packets to be routed in certain ways for diagnostic or other reasons. Multiple naming also allows the use of nicknames for user convenience. Many of these same comments would apply to hosts attached to multiple networks.

An interesting addressing and routing problem arises in mobile packet radio networks. Since hosts are free to move about, the network will need to dynamically change the routes used to reach each host. For robustness, it is also desirable that hosts be able to attach dynamically to different packet radios. Thus failure of a packet radio need not prevent hosts from accessing the network. This requires that host names and perhaps host addresses be decoupled from packet radio addresses. The network must be able to search for hosts or alternatively, hosts must "report-in" to the network so that their addresses can be associated with the attached packet radio to facilitate route selection based on host address. This is just a way of supporting *logical host addressing* rather than using the more common *physical host addressing* in which a host's address is an extension of the packet-switch address.

A crucial issue in network interconnection is the extent to which it should or must impact addressing procedures which are idiosyncratic to a particular network. It is advantageous not to require the subscribers on each network to have detailed knowledge of the network address *structure* of all interconnected networks. One possibility is to standardize an internetwork address structure which can be mapped into local network addresses as needed, either by subscribers, by gateways or by both. Subscribers would know how to map internetwork service names into addresses of the form NETWORK/SERVER. Subscribers need not know the fine structure of the SERVER field. Gateways would route packets on the basis of the NETWORK part of the address until reaching a gateway attached to the network identified by NETWORK. At this point, the gateway might interpret the SERVER part of the address, as necessary, to cause the packet to be delivered to the desired host.

The addressing strategy presently under consideration by CCITT (X.121, [30]) is based on the telephone network. Up to 14 digits can be used in an address. The first 4 digits are a "destination network identification code" or DNIC. Some countries are allocated more than one DNIC (the United States has 200). The remaining ten digits may be used to implement a hierarchical addressing structure, much like the one used in the existing telephone network.

Since the CCITT agreements are for international operation, it might be fair to assume that the United States will not need more than 200 public network identifiers. However, this scheme does not take into account the need for addressing private networks. The private networks, under this addressing procedure will most likely appear to be a collection of one or more terminals or host computers on one or more public networks. It is too early to tell how much this asymmetry in addressing between public and private networks will affect private multinetwork protocols.

A related problem which is not unique to network interconnection has to do with addressing (really multiplexing and demultiplexing) at higher protocol levels. The public carriers tend to offer services for terminal as well as host access to network facilities. This typically means that addresses must be assigned to terminals. The issue is whether the terminal address should be associated with or independent of the protocols used to support terminal-to-host communication.

The present numbering scheme would not distinguish between a host address and a terminal address. A host might have many addresses, each corresponding to a process waiting to service calling terminals.

There has been discussion within CCITT concerning "subaddressing" through the use of a user data field carried in virtual call "setup" packets. This notion would support the concept of a single host address with terminal or process level demultiplexing achieved through the use of the user data field subaddressing.

It seems reasonable to predict that, as terminals increase in complexity and capability, it will eventually be attractive to support multiple concurrent associations between the terminal and several remote service facilities. Applications requiring this capability will need terminal multiplexing conventions beyond those currently provided for in the CCITT recommendations.

To simplify implementations of internet protocol software, it is essential to place bounds on the maximum size of the NETWORK/SERVER address. Otherwise, subscribers may have to construct name-to-address mapping tables with arbitrarily large and complex entries.

Even if all these issues are resolved, there is still a question of "source routing" in which a subscriber defines the route to be taken by a particular packet or virtual circuit. Depending on the range of internetwork services available, a subscriber may want to control packet routes. It is not yet clear how such a capability will interact with access control conventions, but this may be a desirable capability if gateways are not able to automatically select routes which match user service requirements.

D. Flow and Congestion Control

For purposes of discussion, we distinguish between flow and congestion control. Flow control is a procedure through which a pair of communicators regulate traffic flowing from source to destination (each direction possibly being dealt with separately). Congestion control is a procedure whereby distributed network resources, such as channel bandwidth, buffer capacity, CPU capacity, and the like are protected from oversubscription by all sources of network traffic. In general, the successful operation of flow-control procedures for every pair of network communicants does not guarantee that the network resources will remain uncongested.

In a single network, the control of flow and congestion is a complex and not well understood problem. In a multinetwork environment it is even more complex, owing to the possible variations in flow and congestion control policies found in each constituent network. For example, some networks may rigidly control the input of packets into the network and explicitly rule out dropping packets as a means of congestion control. At the other extreme, some networks may drop packets as the sole means of congestion control.

At this stage of development, very little is known about the behavior of congestion in multiply interconnected networks. It is clear that some mechanisms will be required which permit gateways and networks to assert control over traffic influx especially when a gateway connects networks of widely varying capacity. This problem is likely to be most visible at gateways joining high speed local networks to long-haul public nets. The peak rates of the local nets might exceed that of the long-haul nets by factors of 30-100 or more. Generic procedures are needed for gateway/network and gateway/gateway flow and congestion control. Such problems also show up in single networks, but are amplified in the multinetwork case.

E. Accounting

Accounting for internetwork traffic is an important problem. The public networks need mechanisms for revenue sharing and subscribers need simple procedures for verifying the accuracy of network-provided accountings.

The public packet-switching networks appear to be converging on procedures which account for subscriber use on the basis of the number of virtual circuits created during the accounting period and the number of packets sent on each virtual circuit. Indeed, it has been argued that accounting on the basis of virtual circuits at gateways requires less overhead than accounting on a pure datagram basis [32]. Scenarios can be cited which support the opposite conclusion.

Suppose there is a choice between setting up virtual circuits for each transaction and sending a datagram for each transaction, and that virtual circuit accounting includes information on each virtual circuit setup (as in the present telephone network). If datagram accounting simply accumulates the number of datagrams sent between particular sources and destinations without regard to the time at which they are sent, then the amount of accounting information which is collected for the datagram case will be substantially less than for the virtual circuit case. In the limit (i.e., one packet per transaction), the virtual circuit accounting information is proportional to $2N$, where N is the number of transactions, while for the datagram case, it is proportional to $\log N$ (base 2). This is simply because the datagram case only sums counts for traffic between source/destination pairs while the virtual circuit accounting would identify start/stop times for each virtual circuit.

Alternatively, if the bulk of the traffic involves a large number of packets per transaction, then the two accounting procedures would accumulate more nearly the same information since each would predominantly involve accounting for packet flow.

If it is chosen not to account for virtual circuit duration, but merely to account independently for the number of virtual circuits and the number of packets sent between source/destination pairs, then the virtual circuit accounting would be closer to the datagram case.

The important conclusion to be drawn is that accounting for datagrams is generally less complex than accounting for virtual circuits, but that the two can be made arbitrarily similar by suitable choice of the details of the accounting information collected.

F. Access Control

In multinetwork environments, it may be necessary for each network to establish and enforce a policy for "out-of-network" routing. For example, a public network might conclude agreements with other networks regarding the type and quantity of traffic it will forward into other networks. This might even be a function of the time-of-day. Consequently, mechanisms are needed which will permit networks to prevent traffic from entering or leaving or to meter the type and rate of traffic passing into or out of the network.

Another example of the need for control arises with the possibility of third-party routing. That is, traffic destined from network A to network B is routed through network C. It cannot be assumed that all networks have gateways to all others. However, some nets may want to limit the amount of *transit* traffic they carry. There may be explicit agreements among a subset of the nets regarding revenue sharing for transit services. If a particular network does not have a revenue-sharing agreement with the particular source/destination networks of a given virtual circuit or datagram, then it must be able to reject the offending traffic if it so chooses.

There does not seem to be any technical barrier to separating the access control policy decision mechanism from the enforcement of the policy. For example, a gateway might simply enforce policy by sending traffic for which it has no known access rules to an *access controller*. If we adhere to the model that gateways have two *halves*, then each half deals with the network to which it is connected. The access controller can either dynamically enable the flow by causing table entries at the gateways which permit the flow to be created or it can tell the gateway to reject all further traffic of that type.

Clearly, access control policies will affect routing strategies, so this adds a complicating factor into any internetwork routing strategy implemented by the gateways. At present, very little experience has been accumulated with internet access control and routing policies. For the most part, agreements among public networks have been bilateral and transit routing has been treated as a very special case. When EURONET [6] becomes operational, this problem will be particularly important to solve.

G. Internet Services

It is by no means clear what set of services should be standardized and available from, at least, all public data networks. The current CCITT recommendations provide for virtual circuit service and terminal access service on all public packet-switching networks.

Although the recommendations (X.3, X.25) provide for *fragmentation* of packets being delivered to a subscriber on a virtual circuit, the current X.75 gateway draft recommendation uses an agreed maximum packet size of 128 octets of data, not including the header. This agreement avoids for the moment the need to fragment packets crossing a network boundary, as long as all subscribers recognize that the maximum length internetwork packet allowed is 128 octets. Bilateral exceptions to this rule may develop but neither a fixed size nor a collection of special cases represent a very general solution to this problem.

It has been argued [25] that a general scheme for dealing with fragmentation is desirable so that new network technologies supporting larger packet sizes can be easily integrated into the multinetwork environment.

Apart from fragmentation, there are a set of special services such as multidestination addressing and broadcasting which could be used to good advantage to support multinetwork applications such as teleconferencing, electronic mail distribution, distributed file systems, and real-time data collection. Other services such as low delay, high reliability, high bandwidth, and high priority are also candidates for standardization at the internet level.

As in the case of access control, selection of such services might constrain the choice of packet routing to networks capable of supporting the desired services. Once again, very little experience with standard internet services has been accumulated so this subject is still a topic for research. For the most part, terminal-to-host services have been successfully offered across network boundaries using nearly all of the network interconnection methods described in this paper. It remains to be seen whether more complex applications can be equally well supported.

VII. X.25/X.75–The CCITT Strategy for Network Interconnection

The common network access interface concept is favored by CCITT for network interconnection. In the CCITT model of packet networking, all networks offer the same interface to packet-mode subscribers and this is called X.25. X.25 is a virtual circuit interface protocol. However, gateways between networks employ an interface protocol called X.75 [33], which is much like X.25 but accommodates special network/network information exchange, such as routing information, accounting information, and so on.

Fig. 10(a) illustrates the basic network interconnection strategy proposed by CCITT. To appreciate the difference between this strategy and the "common subnetwork" strategy, it is necessary to have some understanding of the X.25 packet network interface. X.25 provides a virtual circuit interface for the setup, use, and termination of virtual circuits between subscribers of the networks. X.25 provides for flow control of packets per virtual circuit flowing into or out of the network. Subscribers may set up switched virtual circuits by sending "call request" packets into the network and receiving "call confirmation" packets in return. The standard also provides for permanent virtual circuits.

The public networks plan to employ X.25 interfaces; it can therefore be assumed that source and destination hosts in different networks will essentially want to exchange "call request" and "call accepted" packets through the mediation of one or more gateways. This strategy could result in a series of virtual circuits chaining source host to gateway, gateway to gateway, and gateway to destination host; alternately an end-to-end virtual circuit could be set up from source host to destination host, with the gateways acting as relays without any special knowledge of the virtual circuits passing across the network boundary.

The principle difference between the X.25 interface and X.75 interface is that virtual circuit setup and clearing packets are passed transparently by the X.75 gateway to the next gateway or destination. For reasons which are described below, it is necessary to maintain the sequence of packets belonging to a given X.25 virtual circuit as they pass through a gateway and enter the next network. Therefore, a virtual circuit is in fact created between the source host and intermediate gateway and between gateways. The X.75 gateway does not spontaneously generate any "call acceptance" packets in response to "call request" packets, but it does participate in the sequencing and flow control of packets on each virtual circuit passing through. Other differences between the X.25 and X.75 interface have to do with the nature of the internetwork accounting or routing information which might be exchanged over X.75 which would not be appropriate for a subscriber to exchange with the network over the X.25 interface.

The design of the X.75 type of gateway depends in principle upon all networks' use of the X.25 subscriber interface. Some networks, like the ETHERNET, cannot implement it without extensive modification, because there are no packet switches in the network to support the required packet reordering at the destination. The alternative is to insist that all internet applications rely on a sequenced data protocol built into the hosts or front-ends. For some services, such as packet speech, the potential overhead of resequencing packets before delivery to the destination may prevent the service from being viable. This problem could be amplified if packets are constrained to remain in sequence as they pass the X.75 boundary.

Fig. 10(b) and (c) shows variants of the CCITT interconnection strategy. In Fig. 10(b), we see an example in which only X.25 is used both as a network access method and as a means of passing traffic across network boundaries. A single subscriber or a pair of subscribers to two nets could interface to their networks via X.25 and to each other by means of some agreed and possibly private protocol.

Virtual circuits would be explicitly set up from source host to gateway, gateway to gateway, and gateway to destination host. The "internet" addresses of the source and destination hosts could be carried in the so-called "Call User Data Field" of an X.25 Call Request packet. This leaves the packet address field free to identify intermediate destinations (e.g., gateways), but preserves an ultimate internetwork source/destination address which the gateway can use to select the destination to which the next intermediate virtual circuit is to be set up.

An alternative to this is shown in Fig. 10(c) in which the subnets *A* and *B* use nonstandard virtual circuit interfaces, but agree to build gateway software employing X.75 signaling procedures across the gateway interface. This solution is substantially the same as that shown in Fig. 10(b), except there is now additional translation software in each gateway half to make each virtual circuit network-access protocol compatible with X.75 procedures.

There are some specific problems with the X.25/X.75 gateway strategy, which do not necessarily apply to other virtual call gateways [63]. The basic X.25 interface provides for the sequence numbering of subscriber packets mod. 8 or, optionally, mod. 128. Since X.25 is an interface specification, this numbering can only be relied upon to have local significance (i.e., host-to-packet switch). Some X.25 implementations use these host-assigned sequence numbers on an end-to-end basis. Others generate internal, network-supplied numbers to allow for repacking of subscriber packets into larger or smaller units for transport to the destination. If packet sequence numbers assigned by the source host were carried transparently to the destination without change, it might be possible to allow packets to flow out-of-order across the X.75 boundary to a gateway and thence into the next network. If the packet sequence numbers were still intact, they could be carried out-of-order to the next destination which might either be a gateway or an X.25 host. In the latter case, the original packet-sequence numbers could be used to resequence the packets before delivery. If the packets were being delivered to an intermediate gateway, they would not have to be sequenced there. However, the X.25 interface specification does not undertake to carry the host-supplied sequence numbers to the destination gateway or host in a transparent fashion, primarily so that the subnetwork can deal more freely with the physical packaging of the packet stream. For example, a source may supply packets of length 128 bytes while a destination may prefer to receive packets no longer than 64 bytes. To allow for such variations, the network must be free to renumber packets for delivery. These considerations have two consequences.

1) X.25 packet sequence numbers cannot be relied on for end-to-end signaling, though they could be so used if requisite information is known about the intermediate transit networks.

2) Packets must be delivered in sequence when passing to or from gateways and hosts on X.25 networks.

The second conclusion may be modified slightly. It is at least essential that packets be delivered in relative sequence on each virtual circuit. By maintaining independent sequence numbering on each virtual circuit, it is possible for hosts and gateways to refuse traffic on one virtual circuit while accepting traffic on another. There are two penalties for this. First, a gateway must keep track of which virtual circuits are passing through it. Second, dynamic alternate routing of packets belonging to the same virtual circuit through alternate gateways is not possible without resetting or clearing the virtual circuit. This last point is simply the consequence of not defining an end-to-end sequence numbering scheme, but instead relying on sequencing of the packets of a virtual circuit on entry to and exit from each intermediate network.

Some networks implement X.25 level acknowledgments (i.e., level 3) that have an end-to-end significance, but others make this purely a host-to-packet switch matter. As a conse-

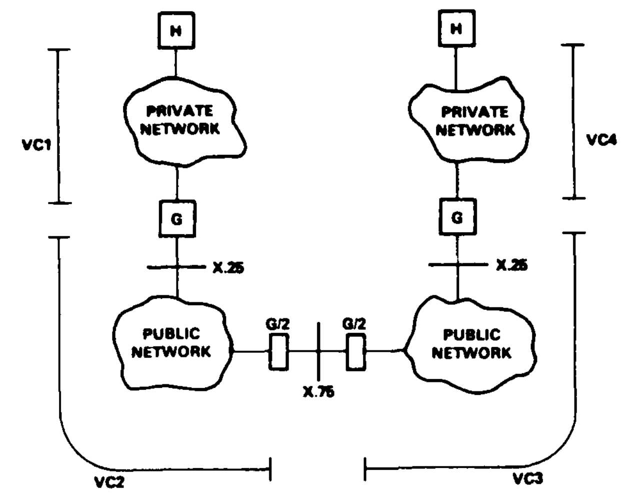


Fig. 14. Use of X.25 for public/private network interconnection.

quence, it is not possible to rely on X.25 packet acknowledgments to determine which, if any, packets were not delivered as a result of the resetting or clearing of a virtual circuit. Furthermore, even if a subnet were to offer an end-to-end acknowledgment between a source host and an X.75 gateway, this could not be assumed to guarantee that the acknowledged packet was delivered to the ultimate X.25 destination in another network.

X.75 is an interface intended for use between public networks. Thus, it is not likely to be used or even allowed as an interface between public networks and private networks. For the case illustrated in Fig. 14, X.25 interfaces could be provided between public and private networks (or other special interfaces) and X.75 interfaces between public networks. Consequently, gateways between public and private networks are likely to appear to be ordinary host computers in the view of the public networks.

The use of X.25 for private/public network interfaces and X.75 for public/public network interfaces leads to the situation shown in Fig. 14 in which an internetwork virtual circuit would have to be made up of several concatenated parts such as virtual circuits 1-2-3-4 (see also [52, Fig. 3.4]). Even if X.25 implementations uniformly permitted an end-to-end interpretation of packet sequence numbers and acknowledgments, there would still be separate virtual circuits required between the source or destination hosts and the gateways into the public networks. However, the concatenation of virtual circuits does not yield a virtual circuit. For instance, a gateway between the public and private net could acknowledge a packet but fail to get it delivered, in which case the subscriber will have been misinformed as to the delivery of the packet. This situation forces the end subscribers of private networks to implement end-to-end procedures on top of any concatenated virtual circuits provided by the public networks.

VIII. Practical Network Connections and Experiments in Progress

A number of networks have been connected successfully over the last few years. Most of these connections have been made in an *ad hoc* manner, using one of the following techniques.

1) One network is a star network with remote RJE and interactive stations. The other is a star or distributed network with clearly defined protocols. A device on the star network provides exactly the functions required by its own network on one side, and those of the other network on the other side.

2) Formal gateways are provided between the two networks, and protocol mapping occurs in the gateway.

3) A computer is a host on two networks. It is arranged that services are provided by accepting input from one network and putting it out on another, possibly after substantial processing.

4) Formal gateways are provided between the two networks. Sufficient agreement is obtained that end-to-end protocols (even high level ones) are common in the two networks. In this case, less activity is required in the gateway.

In the first method, a form of front-end computer is used. It has been adopted in the large airline and banking networks SITA [13] and SWIFT [14]. In each case the standards for the networks have been defined rigidly. SWIFT has even certified officially t[illegible] devices of three manufacturers to provide interfaces to its network. The other side of the device is then programmed to meet the requirements of the star system being attached. In the two cases cited, only a simple message level of interface needed to be defined.

Other examples of the same technique are the connection of the Rutherford Laboratory (RL) star system [53] and the Livermore CTRNET to ARPANET. In these examples, more serious protocol mapping was required. ARPANET has a well-defined set of HOST-IMP, HOST-HOST, Virtual Terminal, and File Transfer protocols. All these had to be mapped into the appropriate procedures for the other network.

The second method has been applied only experimentally. The UCL interface between ARPANET and the UK Post Office Experimental Packet Switched Service (EPSS, [55]) and the National Physical Laboratory interface between EPSS and the European Informatics Network (EIN, [56]) are examples of this technique; a demonstration has even been made of EIN-EPSS-ARPANET with no extra problems encountered from the three networks being concatenated. Technically there is almost no difference between the first two methods. The second looks at first sight somewhat more general than the first, but almost the same problems have to be overcome. The difficulties come from the fundamental differences in the design choices made in the protocols of the different networks; these differences are in general difficult, and even sometimes impossible, to resolve completely. In the first method, they can sometimes be resolved using a specific facility in the star network; in the second, where two distributed networks are involved, this recourse may no longer be available.

One example of the problem occurs in the connection of EPSS and ARPANET. ARPANET can forward any number of characters at a time, and often uses full duplex remote echoing. EPSS works in a half-duplex mode, forwarding only complete records. A special "Transmit Now" has to be input by the user, and interpreted by the gateway, to ensure that partial records are forwarded. Another example, from the same application, occurs in File Transfer. ARPANET assumes an interactive process is live throughout the file transfer; all completion codes are passed over this live channel. The RL network (and EPSS) assume that file transfer is a batch process; they return network completion codes at a later time, and may delay acting on the commands. With the ARPANET-RL link [53], the file transfer job had to be given a very high priority, so that the completion code usually arrived before a timeout occurred; because of the nature of the way the computer was used for large real-time jobs, this did not always ensure that the job was run in a reasonable time.

There are several examples of the third technique. A DEC PDP 10 machine used on the Stanford University SUMEX project is a host both on ARPANET and on TYMNET; several machines at Bolt, Beranek and Newman are both on ARPANET and TELENET. Because the TENEX operating system has good facilities for linking between programs, it would be possible for interactive streams to come in one network and go out on another. File transfer problems would be simple in this configuration, because the hosts obey all the conventions, in any case, of each network. Of course, this mode of operation may require that files in transit between networks may have to be stored temporarily in their entirety in the host serving as the gateway between the networks.

The fourth technique is newer, and has many variations. As a result of agreement on the X.25, and partial agreement on the X.75, protocols, PTT networks are able to interconnect in a reasonably straightforward manner. The connections between DATAPAC and both TELENET and TYMNET have been done in this way. In each case, there has not been any agreement on higher level protocols, so the problems of host-host communication across concatenated networks is not resolved by these linkups of the subnets.

The ARPA-sponsored INTERNET project has tried to standardize to a higher level. A host-host protocol has been defined (TCP, [25]), and is being implemented on a number of different networks including Packet Radio [20], [21], ETHERNET [18], LCSNET [64] and the SATNET [22], in addition to ARPANET. This protocol is defined for use across networks; thus each packet includes an "Internet Header" which is kept invariant as the packet crosses the different networks. One aspect of the INTERNET program is to develop gateways which can interpret this header appropriately.

By late 1976, the ARPA project had connected together the Packet Radio Network, the ARPANET, and the Atlantic Packet Satellite Network using two gateways between the Packet Radio Network and the ARPANET and three gateways between the ARPANET and Packet Satellite Network. It is routinely possible to access ARPANET computing resources via either of the other nets and to artificially route traffic through multiple nets to test the impact on performance. In one such test, a user in a mobile van in the San Francisco area accessed a DEC PDP-10 TENEX system at the University of Southern California's Information Sciences Institute over the following path:

1) from van to the first gateway into ARPANET via the Packet Radio Network;
2) across the ARPANET to a second gateway in London, using a satellite link internal to the ARPANET;
3) across the Atlantic Satellite Network to a third gateway in Boston;
4) across the ARPANET again to USC-ISI.

The user and server were 400 geographical miles apart, but the communication path was 50000 miles long and passed through three gateways and four networks. Except for a slightly increased round-trip delay time, service was equivalent to a direct path through the ARPANET. Since the Packet Radio Network is potentially lossy, can duplicate packets, and can deliver packets out of order, the end/end TCP protocol was used to exercise flow and error control on an end-to-end basis. The availability of a common set of host-level protocols substantially aided the ease with which this test could be conducted.

The ARPA project also has high-level standard protocols already in existence to support file transfer and virtual terminals (the FTP and TELNET protocols [40]), and these are being retrofitted above the internet TCP protocol to provide a standard high-level internetwork protocol hierarchy.

IX. Regulatory Issues

The regulatory issues in the interconnection of packet networks takes a different form in North America than elsewhere. It is hard in a paper of this type to more than touch on some of the problems involved. The discussion here is simplistic in the extreme, and no attempt is made to put the issues in the legalistic language they really require.

In almost all countries the provision of long distance communication transmission and switching is provided by a regulated carrier. In most countries outside North America, this carrier is a single national entity—called the "PTT". In some countries (e.g., Italy) there are different carriers for different services—e.g., telegraph, telephone, intercity, international telephone, etc. In North America there are many carriers. Usually only one in each geographical area has a monopoly on public switched voice traffic. Also the so-called "Record Carriers" have some sort of monopoly on "record traffic," which is message traffic. In a "Value Added Network" (VAN), the operators rent transmission equipment from the carriers, and then add their own switching equipment. These VAN's are themselves regulated in what they may do, what traffic they may carry, and what rates they may charge. Between North America and Europe, specific "International Record Carriers" (IRC) have monopoly rights on data and message transmission —in collaboration with the appropriate European PTT's. The regulations take into account who owns the hosts and terminals, who owns the switches, who rents the transmission lines, what types of traffic is carried, what is the geographic extent of the network, and what is the technology of long distance transmission.

In Fig. 15, a single network N is sketched. It consists of switches S and transmission lines L; these together are called the data network, DN. It consists also of terminals T and hosts H; the exact difference between a terminal and a host is not very clear; we believe it is assumed that terminals mainly enter and retrieve data without processing; while a host transforms the information by processing. This definition probably does not meet the picture of modern "intelligent terminals," but it is always hard for the regulations to keep up with the technology. If the total network is all localized in one site, so that no communication lines cross public rights of way, then it can usually be considered from a regulatory viewpoint, as a single host in more complex network connections. The hosts and the terminals can be connected to the switches, and the switches to each other, either by leased lines, or by the Public Switched Telephone Network; the first type of connection is called a *leased* connection, the second *switched*. In the subsequent discussion of this section, the term "host" will include localized networks. In general we will assume the connections between the switches are via leased lines; if that is not the case, the regulations are much eased in general (though in some countries, like Brazil, no data transmission is permitted at all via switched telephone lines).

If all the hosts and switches are owned by one organization P, which also leases the lines, then P is said to own and operate the network, and it is called a "Private Network." There are

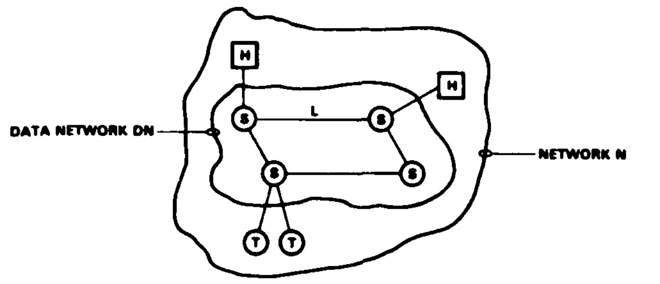


Fig. 15. Schematic of one network.

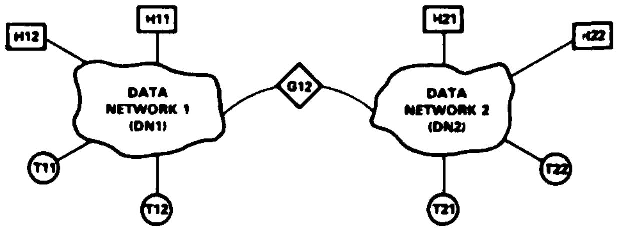


Fig. 16. Schematic of two connected networks.

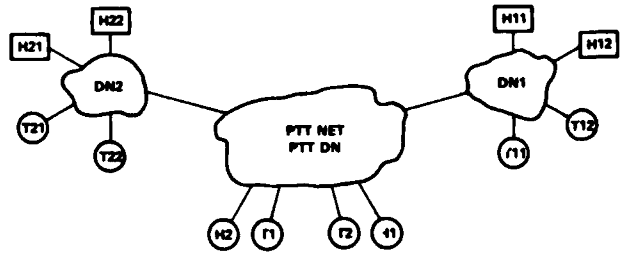


Fig. 17. Schematic of PTT model.

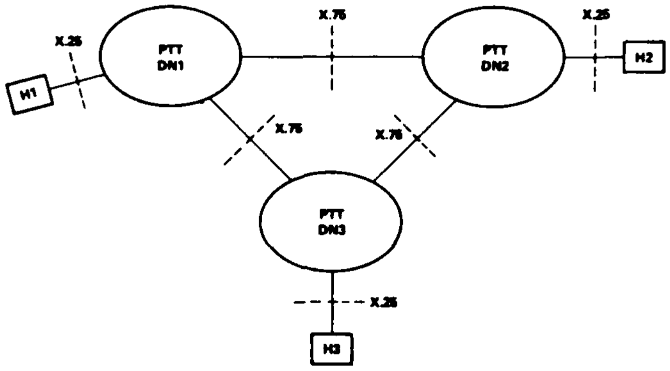


Fig. 18. Multiple PTT network interconnection.

minimal restrictions on such networks—though in West Germany, for example, higher tariffs are charged for the leased lines if any terminals or hosts are connected via the PSTN. In most countries such a network may not be used for the transfer of messages between terminals belonging to organizations other than *P*.

If the data network belongs to one organization, and the hosts to others, the data network is a VAN. Stringent regulations apply to VAN's, in most countries. With rare exceptions, in most European countries, VAN's can be operated only by the PTT's. In the U.S., they can be operated by other organizations, but only if approved as regulated Value Added Carriers (VAC's) by the Federal Communications Commission (FCC). One regulation imposed by the U.S. is that an organization operating as a VAC may not also operate a host for outside sale of services. For this reason, the companies TYMSHARE and ITT have had to spin off their VAC's into separate subsidiaries, TYMNET and ITT Data Services.

In the past, a few VAN's have been permitted to operate internationally for specific interest groups. Two such VAN's are SITA [14], for the airlines, and SWIFT [14] for the banking community. Here the regulations can be stringent. SWIFT has to pay specially high tariffs for its leased lines; its license to operate may be revoked when the PTT's can offer a comparable international service.

As soon as two networks, owned by different organizations, are interconnected, there are regulatory difficulties. This situation is illustrated schematically in Fig. 16. Even if one network is an internal one, so that it can be treated as a single host, its connection to other network immediately changes the latter's status. Thus in Fig. 16, the connection of *DN*1 to *DN*2 immediately changes *DN*2 to a VAN. In Europe it has been decreed that such private networks may not connect directly to each other, but only through a PTT network. Thus the most general configuration permitted by the European PTT's is illustrated in Fig. 17. Moreover, the PTT's have also agreed that only the X.25 interface will be provided to customers, though that interface was defined for the configuration of Fig. 15 rather than 17. The different PTT networks will themselves connect to each other by the different interface X.75 as illustrated in Fig. 18. This does not change, however, the interface seen by the private networks. Further work is needed to assess the suitability of X.25 in this role.

In the U.S., the regulations are not quite so stringent. Connections such as Fig. 15 are permitted even where one host belongs to a different organization than the network operator *P*—provided such connection is only limited and for the purposes of using the facilities of that network. This type of relaxation is really necessary, because of the difficulty of distinguishing between a "host" and a "terminal". In practice, in most countries, the line is drawn between leased line and PSTN connections. The former are usually not permitted without change of status of the network; the latter seem to downgrade the connection to that of a terminal.

The discussion above has treated the types of connections which can be made. In addition, the PTT's, and the FCC in the U.S., usually regulate the purposes for which the network can be used. In particular, there is a ban on such networks being used for message or voice transmission between organizations. How such measures are to be policed, gets us into another regulatory problem. For example the UK PO [57] has claimed a right to inspect the contents of any data message sent across lines leased from it; this right would be at variance with the privacy laws being enacted in many countries [58], [59]. This subject is a large one in its own right, and it is clearly beyond the scope of this paper.

Two other service problems will arise in international connections. First the impact and form of the privacy and transnational data flow regulations in different countries are different. Thus in the interconnection of international networks, a particular set of problems may arise, even when the appropriate regulations are obeyed in each network separately. Thus both Network 1 in country *A* and Network 2 in country *B* may obey their own national regulations. However when the networks *A* and *B* are connected, Network 1's practices may break country *B*'s regulations, and yet be accessible from country *B*. It is this class of problems which delayed seriously the permission by the Swedish Data Inspectorate Board for Swedish banks to connect their networks to SWIFT.

Secondly, some of the functions of networks or gateways legal in one country may be illegal in another. Thus U.S. carriers are not permitted to do data processing in their data networks; no such considerations apply in most European countries. Some of the protocol translation activities, some of the message processing activities, and some of the high-level services (e.g. the provision of multiaddress links) may well be classed as "Data Processing," and hence be illegal in the U.S. In interconnected networks, this raises the possibility that functions can be carried out outside the jurisdiction of the country in which the operator initiating the activity is sited, and yet which is illegal in that country. This subject is treated rather fully elsewhere [60]. A clear example of this is the use of message services operated by TYMSHARE and CCA on TELENET and TYMNET. While these services are legal in the U.S., their use by UK persons connected to TYMNET by the official International Packet Switched Service is clearly technically illegal; this use would contravene the UK Post Office Monopoly.

X. Unresolved Research Questions

There are many unresolved research questions; on some of them even the present authors do not agree with each other! Primarily these questions have a technical, policy, administrative, economic, regulatory, or operational aspect, or a combination of these.

One example of this is the question of the procedures to be used for internet routing. Here there are technical questions on what is feasible in view of the technologies used in the subnets; there are policy questions on when third country routing might be allowed; there are economic considerations on how much it would cost to do the necessary protocol translation to route through third countries, and on what charges the connecting transit network might make; there may be regulatory questions on which classes of data may flow through specific countries (related to the transnational data flow regulations); and there may be operational questions on whether in the event of failure in dynamic rerouting, reestablishment could take place with sufficient rapidity.

Among the outstanding research questions are, in alphabetic order, the following.

Access Control: What are the requirements and methods of implementation of access control? How should they affect internetwork routing?

Addressing: How should the International Numbering Plan, which goes to the level of known subscribers of public networks, be extended? Should this extension be in the numbering plan itself, or should additional user and network information be supplied? Should there be local, or only physical, addressing? Should there be internetwork source routing implied by the addressing?

Broadcast Facilities: What is the role of broadcast communication facilities in the provision of internet services? Should facilities using it be offered? Should technologies supporting it use it, particularly at gateways? What are the implications on protocols, especially with respect to duplicate and error detection?

Datagram versus Virtual Call Facilities: How should datagram and virtual call facilities be interconnected? How can one compare the relative performance and costs of the implementations? What criteria should be used in any comparison? When might datagram, or alternatively virtual calls, be desirable or essential between networks?

Data Protection: What are the effects of end-to-end data encryption on protocol translation?

Flow and Congestion Control: To what extent should one adopt congestion and flow control between gateways and their feeding networks, between gateways directly, or between gateways and the source? What are the relative effects of just discarding packets in gateways, and relying on the end-to-end protocol to detect and compensate for this? How is charging for discarded packets arranged?

High Level Protocols: There are still many questions on what should be standardized, and how rigid the standards should be. To what extent should the individual networks support common standards, and to what extent should protocol translation be feasible technically or attractive economically? What are the costs of maintaining standards or the economic advantages of standard hardware and software? How does the technology of individual networks and the proportion of internetwork traffic affect the decisions?

Internetwork Diagnosis: There are many technical problems in isolating faults in concatenated networks. There are also organizational and economic problems on who should be responsible for their repair, and how costs for service failures should be allocated.

Performance: How do choices of design parameters, and network services, affect the costs of the individual networks? How do the individual network performances and costs scale to large networks? How do the choices affect the feasibility, costs and performance of the gateways? How do the variations in technology or choice of parameters affect the performance in interconnected networks?

Routing Policies: To what extent and when should adaptive routing be used between networks? How can one recover from the partitioning of a single network, when there are still routes existing by going through other networks? How should administrative considerations affect routing policies between networks (privacy regulations, economic considerations of internet payments, desire to provide for high availability, etc.)? When is a hierarchical organization more efficient that a direct route search?

Services: What services are needed on an internetwork level? Clearly interactive and bulk transport services must be supported. What else is needed? Should the internetwork facilities be able to support voice, telemetry, and teleconferencing? What is the cost of supporting these latter services, and what is their effect on other facilities?

X.25 and X.75 and Related Recommendations: Is X.25 suitable for transaction processing? Are the present datagram proposals adequate? How should X.25 be extended for internet addressing? How should X.25/X.75 be modified to allow the connection of private to public networks, or private networks to each other? Do the X.3, X.28, X.29 pad concepts extend well to the internet environment, or should they be modified?

XI. Conclusions

In view of all the unresolved questions discussed in Section X, most of the conclusions which can be drawn in this paper must be tentative. From the early part of the paper, we have shown that it is essential that techniques be developed for connecting computer networks. Moreover, no single set of techniques will fit all applications.

The services which will normally have to be supported are terminal access, bulk transfer, remote job entry, and transaction processing. The quality and facilities of the services required will be very dependent on the applications.

The connections between networks can be made at the level of the packet switches or of hosts, and can be on a datagram or virtual call basis. Connection at the packet-switch level requires broadly similar network access procedures, or complex protocol transformation at the gateways between the networks. If the network protocols are different, interconnection can be most easily achieved if done at the host level. The higher levels of service can be mapped at service centers, which need not be colocated with the gateways—but very different philosophies of network services can be very difficult to map. Alternatively, subscribers can implement common higher level protocols if these can be agreed upon.

The principal problems in connecting networks are much the same as those in the design of the individual networks of heterogeneous systems—but the lack of a single controlling authority can make the multinet design problem more difficult to solve. It is essential to resolve the usual problems of flow control, congestion control, routing, addressing, fault recovery, flexibility, protocol standards, and economy. The public carriers have attempted to resolve many of these problems; particularly in the areas of flexibility, addressing, and economy we feel their solutions are not yet adequate. At the higher levels of protocol, much more standardization is required before we have really satisfactory long term solutions.

The advent of international computer networks, private networks which must communicate with other private networks (even if via public ones), and the new applications of computer networks, raise regulatory and legal issues which are far from resolution.

Many technical solutions to the problems of the connection of networks are discussed in this paper. Their applicability in view of the different technical, economic, and policy constraints imposed in different countries must still be assessed.

REFERENCES

[1] L. G. Roberts, "Telenet: Principles and practice," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, pp. 315-329, 1975.
[2] W. W. Clipsham, F. E. Glave, and M. L. Narraway, "Datapac network overview," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 131-136, 1976.
[3] J. Rinde, "TYMNET: An alternative to packet switching technology," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 268-273, 1976.
[4] R. E. Millstein, "The national software works: a distributed processing system," in *Proc. ACM Nat. Conf.*, Seattle, WA. 1977.
[5] A. Danet, R. Despres, A. Le Rest, G. Pichon, and S. Ritzenthaler, "The French public packet switching service, The TRANSPAC network," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 251-269, 1976.
[6] G. W. P. Davies, "EURONET project," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 229-239, 1976.
[7] R. Nakamura, F. Ishino, M. Sasaoka, and M. Nakamura, "Some design aspects of a public switched network," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 317-322, 1976.
[8] F. A. Helsel and A. J. Spadafora, "Siemens system EDS—A new stored program controlled switching system for telex and data networks," in *Proc. Third Int. Computer Communications Conf.*, Toronto, Canada, pp. 51-55, 1976.
[9] T. Larsson, "A public data network in the nordic countries," in *Proc. Third Int. Computer Communications Conf.*, Toronto, Canada, pp. 246-250, 1976.
[10] P. T. Kirstein, "Planned new public data networks," *Comput. Networks*, vol. 1, no. 2, pp. 79-94, 1976.
[11] P. T. F. Kelly, "An overview of recent developments in common user data communications networks," in *Proc. Third Int. Computer Communications Conf.*, Toronto, Canada, pp. 5-10, 1976.
[12] —, "Public packet switched data networks," this issue, pp. 1539-1549.
[13] P. Hirsch, "SITA rating a packet-switched network," *Datamation*, vol. 20, pp. 60-63, 1974.
[14] G. Lapidus, "SWIFT network," *Data Communications*, vol. 5, no. 5, pp. 20-24, 1976.
[15] L. G. Roberts and B. D. Wessler, "The ARPA network," in *Computer-Communications Networks*, N. Abramson and F. Kuo, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1973, pp. 485-500.
[16] P. M. Karp, "Origin, development, and current status of the ARPANET," in *Proc. COMPCON73*, San Francisco, CA, Feb.-Mar. 1973, pp. 49-52.
[17] L. Pouzin, "Presentation and major design aspects of the CYCLADES computer network," in *Proc. Third Data Communications Symp.*, Tampa, FL, Nov. 1973, pp. 80-85.
[18] R. M. Metcalfe and D. R. Boggs, "ETHERNET: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, no. 7, pp. 395-404, July 1976.
[19] A. S. Fraser, "SPYDER—A data communications experiment," *Comput. Sci. Tech. Report*, no. 23, Bell Laboratories, Dec. 1974.
[20] R. E. Kahn, "The organization of computer resources in a packet radio network," in *Proc. Nat. Computer Conf.*, AFIPS Press, pp. 177-186, May 1975.
[21] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman, "Advances in packet radio technology," this issue, pp. 1468-1496.
[22] I. M. Jacobs, R. Binder, and E. V. Hoversten, "General purpose satellite networks," this issue, pp. 1448-1467.
[23] D. Lloyd and P. T. Kirstein, "Alternate approaches to the connection of computer networks," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 499-504, 1975.
[24] L. Pouzin, "A proposal for interconnecting packet switching networks," IFIP Working Group 6.1, General Note no. 60, Mar. 1974.
[25] V. G. Cerf and R. E. Kahn, "A protocol for packet network interconnection," *IEEE Trans. Commun. Technol.*, vol. COM-22, pp. 637-641, 1974.
[26] C. Sunshine, "Interconnection of computer networks," *Comput. Networks*, vol. 1, 1977, pp. 175-195.
[27] CCITT, "Recommendation X.3: International user facilities in public data networks," *Public Data Networks, Orange Book*, vol. viii.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 21-23, 1977.
[28] CCITT, "Recommendation X.25: Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DEC) for terminals operating in the packet mode on public data networks," *Public Data Networks, Orange Book*, vol. VIII.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 70-108, 1977.
[29] CCITT, "Provisional recommendations X.3, X.25, X.28 and X.29 on packet-switched data transmission services," Int. Telecommunications Union, Geneva, Switzerland, 1977.
[30] CCITT, "Recommendation X.121—Int. numbering plan for public data networks," Study Group VII, Temporary Document 76-E, Int. Telecommunications Union, Geneva, Switzerland, April 25, 1978.
[31] L. Pouzin, "Virtual circuits vs. datagrams—Technical and political problems," in *Proc. Nat. Computer Conf.*, AFIPS Press, pp. 483-494, 1976.
[32] L. G. Roberts, "International connection of public packet networks," in *Proc. Third Int. Conf. Computer Communications*, Toronto, Canada, pp. 239-245, 1976.
[33] CCITT, "Recommendation X.75—Terminal and transit call control procedures and data transfer system on international circuits between packet-switched data networks," Study Group VII, Temporary Document 132-E, Int. Telecommunications Union, Geneva, Switzerland, Apr. 25, 1978.
[34] D. J. Farber and L. C. Larson, "The structure of a distributed computing system—The communication system," in *Proc. Symp. Computer Communications Networks and Traffic*, Polytechnic Institute of Brooklyn, pp. 21-27, Apr. 1972.
[35] P. Baran, "Broad-band interactive communication services to the home: Part II—Impasse," *IEEE Trans. Communications*, p. 178, Jan. 1975.
[36] R. E. Schantz and R. Thomas, "Operating systems for computer networks," *Computer*, Jan. 1978.
[37] L. Pouzin and H. Zimmermann, "A tutorial on protocols," this issue, pp. 1346-1370.
[38] F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The interface message processor for the ARPA computer network," in *Proc. Spring Joint Computer Conf.*, vol. 36, AFIPS Press, pp. 551-567, 1970.

[39] S. Carr, S. D. Crocker and V. G. Cerf, "Host-Host communication protocol in the ARPA network," in *Proc. Spring Joint Computer Conf.*, vol. 36. Atlantic City, NJ: AFIPS Press, Montvale, NJ, pp. 589-598, 1970.
[40] E. Feinler and J. B. Postel (Eds.), *ARPANET Protocol Handbook*. Network Information Center, SRI International, for the Defense Communication Agency, Jan. 1978.
[41] R. F. Sproull and R. D. Cohen, "High-level protocols," *this issue*, pp. 1371-1386.
[42] S. D. Crocker, J. F. Heafner, R. M. Metcalfe, and J. B. Postel, "Function-oriented protocols for the ARPA computer network," in *Proc. Spring Joint Computer Conf.*, vol. 40. Atlantic City, NJ: AFIPS Press, Montvale, NJ, pp. 271-279, 1972.
[43] S. M. Ornstein, F. E. Heart, W. R. Crowther, H. K. Rising, S. B. Russel, and A. Michel, "The terminal IMP for the ARPA computer network," in *Proc. Spring Joint Computer Conf.*, vol. 40, Atlantic City, NJ: AFIPS Press, Montvale, NJ, pp. 243-254, 1972.
[44] CCITT, "Recommendation X.21: General purpose interface between data terminal equipment (DTE) and data-circuit terminating equipment (DCE) for synchronous operation on public data networks," *Public Data Networks*, Orange Book, vol. VIII.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 38-56, 1977.
[45] CCITT, "Recommendation X.21-bis: Use on public data networks of data terminal equipments (DTE's) which are designed for interfacing to V-series modems," *Public Data Networks, Orange Book*, vol. viii 2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 38-56, 1977.
[46] ISO, "High level data link control (HDLC)," *DIS 3309.2 and DIS 4335*, Int. Standards Org.
[47] V. Cerf, A. McKenzie, R. Scantlebury, and H. Zimmermann, "Proposal for an international end-to-end protocol," *Computer Communication Review*, ACM Special Interest Group on Data Communication, vol. 6, no. 1, Jan. 1976, pp. 63-89.
[48] A. S. Chandler, "Network independent high level protocols," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 583-602, 1975.
[49] —, "A network independent file transfer protocol," EPSS High Level Protocol Group, 1977.
[50] R. E. Kahn and W. R. Crowther, "Flow control in resource sharing computer networks," *IEEE Trans. Commun.*, vol. COM-20, pp. 539-546, 1972.
[51] L. Pouzin, "Flow control in data networks–Methods and tools," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 467-474, Aug. 1976.
[52] G. V. Bochmann and P. Goyer, "Datagrams as a public packet-switched data transmission service," Universite de Montreal, *Departement D'Informatique Report*, Mar. 1977.
[53] P. L. Higginson, "The problems of linking several networks with a gateway computer," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 453-465, 1975.
[54] J. Shoch, *private communication*.
[55] P. L. Higginson and Z. Z. Fischer, "Experience with the initial EPSS service," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 581-600, 1978.
[56] D. L. A. Barber, "A European informatics network: Achievements and prospects," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 44-50, 1976.
[57] B. Cross, "General license for message conveying computers," *London Gazette*, pp. 7662-7663, May 28, 1976.
[58] J. Freese, "The Swedish data act," in *Proc. Conf. Transnational Data Regulation*, Brussels, Belgium, ONLINE, pp. 197-208, 1978.
[59] R. Turn, "Implementation of privacy and security requirements in transnational data processing systems," in *Proc. Conf. Transnational Data Regulations*, Brussels, Belgium, ONLINE, pp. 113-132, 1978.
[60] A. R. D. Norman, "Project goldfish," in *Proc. Conf. Transnational Data Regulations*, Brussels, Belgium, ONLINE, pp. 67-94, 1978.
[61] E. Raubold and J. Haenle, "A method of deadlock-free resource allocation and flow control in packet networks," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 485-487, Aug. 1976.
[62] J. McQuillan, "The evolution of message processing techniques in the ARPA network," *Network Systems and Software*, Infotech State of the Art Report 24, Infotech Information Limited, Nicholson House, Maidenhead, Berkshire, England, 1975.
[63] P. Curran, "Design of a gateway to interconnect the DATAPAC and TRANSPAC packet switching networks," *Computer Communication Networks Group*, E-Report E-67, University of Waterloo, Canada, Sept. 1977 (ISSN 384-5702).
[64] D. D. Clark, K. T. Pogran, and D. P. Reed, "An introduction to local area networks," this issue, pp. 1497-1517.

PROTOCOLS IN A COMPUTER INTERNETWORKING ENVIRONMENT

Ray I. McFarland Jr.

United States
Department of Defense

ABSTRACT

This paper presents a model for protocol layering in a computer internetworking environment. Four distinct protocol layers are identified; the network layer, the internet layer, the transport layer, and the application layer. The functions of each are defined. Gateway functions are also addressed in the discussion of the internet layer. A set of protocols are defined for the transport layer based on communications requirements; a reliable data protocol, a datagram protocol, a speech protocol and a real time protocol. Alternatives for standardization at the network, internet and transport layers are presented. Some impacts of choosing each alternative are discussed.

INTRODUCTION

Computer networks are playing a more important role every day within the Department of Defense. More and more projects situated on different networks are finding that they have a requirement to intercommunicate. These requirements, in addition to the direction being taken by DoD to have one long haul common carrier (that is, AUTODIN II) rather than many large geographically dispersed special purpose networks, are leading to the development of computer internetworking strategies.

In order to exchange information in a meaningful way through networks of computers, there must be an agreed upon protocol, or set of protocols. This paper will present a protocol layering model for a computer internetworking environment. Four distinct protocol layers will be identified and their functions defined. The functions of network gateways will also be addressed by the model.

Alternatives for standardization of three of the four layers will be presented. Some of the impacts of the various alternatives will also be discussed.

A PROTOCOL LAYERING MODEL

One of the definitions Webster's New World Dictionary of the American Language gives for protocol is "the code of ceremonial form and courtesies, of precedence, etc. accepted as proper and correct in official dealings, as between heads of states or diplomatic officials" {1}. In much the same way, a communication protocol is a defined set of control procedures and formats for the transmission of information which is agreed to by the owners of the communications gear involved. Protocols can be divided into layers in such a way that each layer implements certain control procedures, which provide a set of communication properties to the layers above it. Ideally, the higher layer protocols should be able to take advantage and build on the properties provided by the layers beneath it.

There are four major protocol layers emerging in the DoD computer internetworking environment. We call them the network layer, the internet layer, the transport layer and the application layer. These four layers are illustrated in Figure 1. For one example which will briefly show how the layers fit together, consider what a message would look like on a network with all four layers present. The first item in the message is the network layer header, which contains the control information for the network layer. Next is the internet layer header, followed by the transport layer header, an application layer header if the application control is not implicit in the data, and finally the data itself. See Figure 2. This section will define the control procedures of each layer.

The ARPAnet will be used in the following discussion to provide examples. Further information on the ARPAnet is given in {2} and {3}. The term 'packets' will be used here to refer to integral units of information transmitted on a network. The term will be qualified, as in 'ARPAnet packets', when referring to specific implementations to avoid ambiguity.

Network Layer

The 'lowest' (furthest removed from the user) layer is the network layer. This layer consists of the control procedures required to actually transmit packets physically between two subscribers on one network (one or both of which could be a gateway to another network), and defines the interface to higher layer protocols. (The concept of a gateway is defined in the Internet Layer section of this paper.) For example, the ARPAnet's network layer consists of the IMP-IMP protocol, (the IMP, which stands for Interface Message Processor, is the ARPAnet packet switch), and that portion of the Network Control Program which implements the Host-IMP protocol, which is usually referred to as the Bolt, Beranek and Newman 1822 Interface Specification {3}. Of necessity, this protocol layer is dependent on the specific network technology. The network protocol for an ARPAnet type packet switching network will be different than one for a ring network, a packet radio network, or a satellite network.

There are two minimal control procedures which all network layers must implement, addressing and routing. As noted in {4} and {5}, these are not the same thing. An address defines where an entity is located and a routing mechanism defines how to get from one address to another. Every network must have the ability to identify the locations of machines on it (i.e., have an addressing scheme). In addition, they must have a scheme for routing packets between two points, whether it is a static or dynamic scheme, predetermined or based on a heuristic algorithm.

There are, of course, additional control procedures which a network layer may provide. One of the most important from a network health standpoint is flow control. A properly implemented flow control scheme allows the network to protect network resources from congestion. Two ways of doing this are throttling network input to a certain maximum level and redirecting traffic around a congestion point with a dynamic routing scheme. For example, the ARPAnet allows only eight ARPAnet messages at a time between any two hosts, while the dynamic routing algorithm was intended (in part) to handle traffic congestion between any two adjacent IMPs.

This layer may also provide error detection, either on a hop by hop basis or on a point of entry to point of exit basis, or some combination of the two. A strictly hop by hop scenario is the strategy typically used i.. a store and forward network. When a switch receives a packet it sends an acknowledgment to the adjacent sending switch, which is then allowed to release its copy. One disadvantage of this scheme is that, if a network malfunction occurs, it is possible to lose messages. A switch crashing after having acknowledged a message but before sending it on is one example. For a strictly point of entry to point of exit scenario, the destination switch would acknowledge the packet to the source switch only after it had successfully passed it to the intended destination. (This is also referred to as end to end acknowledgement.) Thus, if the network malfunctions and drops a packet, recovery is still possible since the source switch has maintained a copy. An acknowledgment from the destination switch had not yet been received by the source switch. The ARPAnet actually uses a combination of the two, with inter-IMP acknowledgments as an ARPAnet packet traverses the network and a Ready For Next Message (RFNM) which is sent from the destination IMP to the source IMP.

A network may provide a form of fragmentation, where messages delivered to the network are broken down into smaller units for transmission. This is another mechanism commonly used by networks to maximize their resources. At the destination switch, the network is responsible for reassembly of the fragments it has created. The ARPAnet breaks messages down into packets for transmission across the IMPs and reassembles the messages at the destination IMP.

A network may also implement some form of precedence strategy for high priority packets.

The overall capability this layer provides is the capability to physically move packets of information between the network's subscribers (or gateways), without requiring the higher layers to have knowledge of the switch procedures or formats.

Internet Layer

This layer consists of the control procedures required to allow internet packets to traverse multiple networks between any two hosts. This protocol is usually implemented within hosts and gateways. The gateway attaches to two or more networks and is the bridge between the networks over which the internet packets flow. The primary function of the gateway is the passing of control information and data between two networks. In addition, the gateway must also determine what network layer control procedures are to be invoked for a particular packet. The gateway derives this information from the internet protocol header. It should not translate between the two network layers. It is preferable to derive the control information needed from the internet header and allow the destination network to implement the required control within the context of its own control constructs rather than try and match up the control constructs of two network layer implementations. In general, the translation of control constructs from one network layer implementation to another is cumbersome and a one-to-one mapping of the control constructs of two network protocols is rarely obtainable. The best chance to achieve such a mapping is if the two network protocols are exactly the same, but even then some 'fudging' of the protocol may be needed for an implementation, (e.g., the implementation and interpretation of a RFNM, which is end to end intranet, but is not end to end internet).

There are three minimal control procedures which the internet layer must implement: addressing, routing and fragmentation. The internet address must be able to uniquely specify a location on a set of networks and also identify the proper transport layer processing software to which the packets should be sent. The usual mechanism for doing this is a hierarchical addressing scheme, such as '<network address><host address><transport protocol processing module address>'. Other addressing schemes have also been devised to try and reduce the overhead of the addressing field in the header. Whatever scheme is implemented, the gateway must be able to map the internet address to a specific network address, either the intended destination host or another gateway.

Gateways determine the routing at the internet layer when more than one gateway must be traversed to reach the destination host. There may even be two different gateways between the destination network and an intermediate network. The gateway between the source network and the intermediate network would then be able to choose which gateway to route the packets through. This can be even more significant when the destination network is at least three networks away from the source network. In this case, the internet routing could actually determine which network(s) the packets are to flow through. For example, network A may attach to network B through one gateway, network B attach to network C through one gateway and to network D through one gateway, network C attach to network E through one gateway, and network D attach to network E through one gateway (see Figure 3). Two alternate routes then exist from network A to network E. One route is A-B-C-E, the other is A-B-D-E. The gateway can adjust to gateway (or intermediate network) congestion by dynamically choosing which gateway individual packets should go through. This is analogous to the dynamic routing algorithm in the ARPAnet mentioned earlier. In the same way that ARPAnet packets of a given message are not constrained to a specific series of IMPs, packets of a given connection should not be constrained to a given series of gateways. However, for this to be possible, the packets of a higher layer protocol connection must not be constrained to go through one specific gateway or series of gateways to reach their destination. (The concept of a connection is defined in the Transport Layer section of this paper.) The gateway should be oblivious to the existence of connections. An additional advantage gained from this approach is the lack of a need for the gateway to store connection state information, allowing for a simple and more efficient gateway. The proper place for connection state information is at the next layer, the transport layer.

The third minimally required control procedure is fragmentation. (Fragmentation in a specific gateway is necessary when one of the attached networks has a maximum packet size which is smaller than one of the other attached networks' maximium packet size. We will assume this as the general case in this discussion.) A gateway must have the capability to interface two networks which have different maximum size packet lengths. To do this, the gateway must be able to break down a packet into fragments, each looking like an integral packet to the network with the smaller size maximum packet length. The internet protocol must, therefore, provide the means for identifying fragments and for sequencing them so that they can be reassembled.

It is important to note that if reassembly of fragments is done at the gateway, then all of the fragments which make up the larger packet are constrained to go through the same gateway when leaving a network. For error recovery by retransmission, the retransmission of the original packet must be constrained to the originally addressed gateway, which may counter any dynamic routing algorithm that may exist at the internet layer. Without dynamic routing, the gateway is a point of single failure for all connections that go through it. With dynamic routing and the requirement for reassembly of fragments at a gateway, the gateway may require some knowledge of the formats and error recovery procedures of all the transport layer protocols which can pass through it. For example, to decide whether to hold or discard a partial packet, the gateway may have to know which transport level protocols retransmit and which do not. This violates the premise that protocol layers should be kept separate and distinct, and not rely on the formats and prcedures of protocols that are at a higher layer. A second disadvantage to dynamic routing with reassembly at a gateway is that a gateway's buffers may be tied up waiting for a 'lost' fragment of a packet while the retransmitted packet has already passed through an alternate gateway.

Where then should the fragments be reassembled? If the reader will recall, we have been discussing the functions of a gateway which are needed to process the internet protocol layer. Yet, nowhere was it mentioned that the protocol layer is either created or terminated at the gateway. The information has already existed for the gateway to process it. All well defined protocols have (at least) two distinct ends, the 'ends' for the internet layer are at the source and destination hosts. The software which implements these internet layer procedures at the hosts could be loosely referred to as 'half a gateway', since it only connects to one network. The source gateway-half is responsible for forming the internet header, deriving the necessary control information from either the host directly or from the transport layer header (e.g., precedence, sequencing information, etc.). The destination gateway-half is responsible for reassembling the fragments and demultiplexing the internet packets to the proper transport protocol processing modules. Of course, some host implementations may not have the capability to reassemble fragments. In this case, the internet protocol must allow for the source host to declare an option of 'do not fragment this packet'. Gateways which have to fragment these type of packets would either discard them or reroute them to another gateway. In fact, this information is one of the things which could go into the routing scheme which gateways implement (including the gateway-half at the source).

For retransmission efficiency, one might wish to trade off some of the flexibility in the previously described dynamic routing scheme for a simple error detection and retransmission procedure between any two gateways. In this case, there is still no need to correlate two different fragments at an intermediate gateway. Individual network packets, when retransmitted from one gateway to the other, would be constrained to go to the gateway addressed originally. However, if the gateway fragments a packet; then new internet checksums are computed for each fragment (which become individual packets for the next network). What is lost is the ability to address the retransmitted packet to an alternate gateway if the gateway addressed originally is overloaded or has crashed. (A more complex procedure could allow for both dynamic routing and gateway error detection and retransmission. The complexity is in the bookkeeping required at the sending gateway to allow it to properly process any returning acknowledgment.)

The internet layer presents the capability to move the packets over many networks and hides the necessary details of gateway functions from the higher layer protocols. For instance, the transport protocol layer need not worry about gateway addressing or network routing.

The functions of a gateway are intimately tied to the internet protocol which it implements; however, the gateway and the internet protocol are not synonymous. A gateway may have other functions beyond the strict implementation of the internet protocol. These functions must, however, meet the requirement that the gateway remain simple, efficient and easily maintainable. One such function has already been mentioned, the maintenance of network congestion information, which contributes to the routing decision at the internet layer. Other functions would be accounting and reporting to some network control center(s) for 'state of the gateway' information, such as queue lengths, traffic density, etc. A gateway could also act as an agent of a network access control center, for network accountability and self protection requirements.

Transport Layer

This layer consists of the control procedures necessary to deliver packets between two application processes on different host computers, whether the hosts are on the same or different networks. Within networks, this layer has been commonly referred to as the Host to Host protocol. The transport protocol modules interface to the application layer software modules (or some host to front end protocol module where the transport protocol is terminated in a front end). It is at the transport layer that explicit connection information makes sense, since connections are thought of as explicitly defining the transmission path between two (or more) application layer processes. Connection state information and connection maintenance is one responsibility of the transport layer protocol.

One type of transport layer protocol is not sufficient for most users. Different users have different communication requirements. These requirements are usually a function of the reliability level needed, timeliness of delivery, and the need for sequenced delivery between two application processes in different hosts. A protocol which attempts to solve all the needs of all users will probably be worthless to everyone. (It will at the least be grossly inefficient, something which cannot be tolerated in a communications environment.)

Four types of protocols seem to be needed at this level; a reliable data protocol, a datagram protocol, speech and a real time protocol. There may be more, but this paper will limit its discussion to these four. The following discussion will not attempt to define all the control procedures a particular transport protocol should have, but will give a sufficient number to allow the reader to distinguish between the four types.

The reliable data protocol is characterized by the need for a high level of reliability and the need for sequential delivery of the packets transmitted between two processes.

The need for sequenced delivery leads to the concept of a communications connection existing between two processes. The defining characteristics of a connection are: (i) each end has an explicit name and is associated with a specific process; and (ii) the packets are sequenced only with respect to the order of transmission on their connection, and independent of the sequencing of packets on other connections. The reliable data protocol is responsible for implementing the concept of a connection. This includes, but is not limited to, the opening and synchronizing of a connection, the maintenance of an open connection and the corresponding connection state information, the resynchronization of a connection if and when necessary, and the closing of a connection. In short, all the connection management functions.

The reliability requirement is satisfied by a mechanism in the reliable data protocol which guarantees packet delivery at the receiver. To do this, the protocol must provide a sufficiently robust error detection scheme (which is usually some form of cyclic redundancy check). The protocol must also provide a way to positively acknowledge packets and must be persistent in the retransmission of packets until, positive acknowledgment of an error free delivery is received or an abort time out period expires. If the abort occurs, the protocol must be able to identify for the user which packets were received and which were not. Since fragmentation, dynamic routing strategies and packets received in error can result in out of order reception of error free packets and possibly duplicate reception of some packets, the protocol must compensate by being able to detect duplicate packets and reorder the original packets prior to delivery to the receiving process. The reordering of packets before delivery also aids in the identification of received packets for an aborted connection.

One last functional requirement for the reliable data protocol is the maintenance of a flow control strategy. At this layer, the flow control strategy is aimed at the management and protection of host resources, such as the amount of buffers available for received traffic.

The second type of transport protocol is the datagram protocol. This protocol is characterized by the lack of a need for sequenced delivery and the decreased level of reliability required.

The datagram 'service' basically has a single packet orientation with no relation existing between packets, something like a telegram service. Becuase of this characteristic, there is really no need for all the overhead involved in the maintenance of explicit connections. In fact, a number of the functions that a reliable data protocol provides, such as flow control, are not even needed. Establishing a connection for a single packet can be a waste of transmission resources and be very inefficient. The datagram protocol must provide a way to identify individual packets, but it does not have to sequence them.

Some individuals within the ARPAnet community who have expressed a desire for a datagram have indicated that their application layer protocol will provide the degree of reliability wanted. An application layer protocol would simply retransmit until some form of positive acknowledgment (either explicit or implicit, such as the results of some initiated action being returned) has been received. The datagram protocol, then, must implement some form of error detection for error free delivery of packets, but it does not have to guarantee the arrival of the packets or reorder them or detect duplicates as the reliable data protocol does.

A speech protocol is a third type of transport protocol. This protocol is characterized by the need for sequenced delivery and the need for very timely delivery.

As in the case of the reliable data protocol, the speech protocol requires a connection management mechanism to preserve logical relationships among the packets through sequenced delivery. Flow control may or may not be required, depending on the particular speech application and available resources.

Individuals who are working on packetized speech have indicated that they would prefer to trade off a highly reliable protocol for one which is very timely in its delivery of packets. (Note that the retransmission of packets received in error or possibly lost in the network reduces the timeliness of their delivery.) Reordering is required for two reasons, ordered delivery to the application process and for the detection of late arriving packets, which are discarded. Though they require reordering, they do not worry about lost or undelivered packets. Gaps in the reordered packets delivered to a speech algorithm do not severly affect the quality of the speech, unless the gap is significantly large. The protocol must then provide for a way of sequencing the packets, reordering them and de cting duplicate fragments. But it does not necessarily have to implement a positive acknowledgment scheme for the purpose of retransmission of packets which were lost or received in error. An error detection scheme is required so that error packets can be discarded at the receiver.

The fourth type of transport layer protocol is for real time traffic. This is the most difficult type of traffic to deal with, since it is characterized by a need for sequenced delivery, and the need for both highly reliable and timely delivery. The distinction made between speech and real time traffic is that with speech, which is ultimately intended for the human ear, all the traffic is not required for intelligent processing of the information. The ear is an excellent filter which can integrate over missing traffic, as long as the gap is not too large. Real time traffic is more in the character of data as described under the reliable data protocol, such as the remote control of a sensitive production process. All of the information is required for processing. The requirements for both high reliability and timely delivery effects the technology choices of the networks over which the information must pass.

The types of functions that this protocol must have is a combination of those defined for the reliable data protocol and the speech protocol.

Transport layer protocols provide the 'transportation medium' to the protocols at the application layer. The transport layer hides from the application layer the implementation details of connection management and flow control, sequencing and packet errors (except for the datagram protocol). Packets generally will be delivered just as they were sent, except as noted earlier.

Application Layer

This layer defines the control procedures between two application processes necessary to accomplish a given task. For example, the control procedures for an information retrieval package might be search, extract, sort, merge, etc.

The application layer protocols provide the capability for two software processes to work together. This layer always exists, whether explicitly or implicitly, whenever two processes are required to communicate, be they on the same machine, the same network, or different networks. When this layer is explicitly defined in the design stage of a project, it increases the understandability of the software requirements and aids in the definition of clean software interfaces.

PROTOCOL STANDARDIZATION

Recalling our initial definition of the word 'protocol', it is interesting to consider what happens when two societies, which have different protocols (or shall we say 'standards' of behavior?) interact. The results can be humorous, confusing, irritating, and sometimes even violent, all at the same time. The effects can be the same when different computer networks, which have different protocols, want to intercommunicate. And this leads to the question of standardization and the degrees of standardization. How much is sufficient? How much is too much? What are the issues? These questions are questions which must be addressed, and they cannot be addressed in a vacuum. What one organization does and how they do it has an impact on other organizations, and vice versa.

Standardization of the Network Layer

The definition of the network layer protocol is primarily a function of the technology of the network because this is the protocol responsible for actually moving packets through the network's physcial switches. The choice of a specific technology for networks is usually driven by the intra-network requirements, as it should be. Real time requirements also play a large role in the choice of network technology.

A network implementer may choose from a number of technologies for his network. Some network implementers might choose an R/F cable (such as the MITRE bus), or radio (such as ARPA's Packet Radio), or the use of commercial land line (such as the ARPAnet), to name just a few. It is, therefore, not really practical to argue for one, or even a few, standard protocols at the network layer. The disadvantages of forcing every network to use the same or extremely similar technologies to meet their requirements far outweigh the advantage of all networks being able to interconnect at the network layer, especially when a strategy exists which allows intercommunication between networks without imposing this type of restriction (one example being the protocol layering model given in this paper).

It does, however, make sense to argue for a standard interface to the higher layer protocol. This would allow relatively easy conversion between two network technologies when a network is upgraded and to some extent allows for transportability of higher layer protocol implementations.

Standardization of the Internet Layer

The internet layer is where the real impact of standardization or the lack thereof occurs.

There are three alternatives for implementing the internet layer: (i) define one standard internet layer protocol to be used within one communication community (such as DoD); (ii) do not standardize at all and allow all networks to implement their own internet layer protocol, requiring a protocol translation at the gateway for the internet protocol; and (iii) do not even have an internet protocol and relegate the functions to either the network layer or the transport layer. Implementation of the internet layer procedures in the network layer protocol now implies that a protocol translation must occur at the network layer. The net effect, from a standardization point of view, is the same as alternative (ii). Implementation in the transport layer protocol falls under the category of standardization for transport layer protocols. The following discussion focuses on alternative (i), a single internet protocol standard and alternative (ii), the lack of a standard.

There are a number of advantages to a standard internet protocol, most of which are reflected in the size and simplicity of the gateway. A standard protocol leads to a common approach for gateway construction, where many copies of the heart of one gateway (the internet protocol implementation) can be made and supplied to many networks. Networks would be responsible for interfacing their particular network layer to the internet layer. These modules should already exist at the network's hosts, where a gateway-half is implemented. This approach can reduce the net development costs for gateways, and software development is an expensive proposition (as we continue to experience). It would also reduce the software maintenance costs. It is possible to have the types of congestion control based on dynamic routing dicussed earlier that would probably not be possible if protocol translation were required, resulting in a form of more reliable service (reliable here in the sense that a gateway is not necessarily a single point of failure or congestion for a user's communication).

The gateway does not have to worry about connection management (which is non-trivial) as it would have to do if the procedures of this layer are relegated to the transport layer, unless a standard transport protocol is implemented. This approach maintains a transparency to all the transport layer procedures. And there may very well be more than one transport layer protocol to worry about. This results in a much simpler, more efficient, and probably more reliable gateway.

On the other hand, one standard internet layer protocol does have its disadvantages. It requires political agreement between orgainzations which is not always easy to obtain, especially when an organization has already invested resources to go in a different direction. Technical conformity is required, something that all skillful protocol designers have trouble living with. And it provides less flexibility to change, at least at the internet layer, to meet new requirements.

When some of the procedures which we feel should be in the internet layer have been relegated to the transport layer, the discussions of the section on transport layer protocol standardization also apply.

Standardization of the Transport Layer

Standardization of the transport layer protocol can also have a significant impact, but not as large as that of the internet layer (unless the internet layer's control procedures are implemented at the transport layer). It is possible to speak about standardization of the transport layer within a community since it is possible to define a set of closed communities which share a common network or set of networks. By a closed community, we mean thrt a host belonging to that community will never talk to a host outside of the community. But, when two closed communities which have their own "standard" transport layer protocols develop a requirement to intercommunicate, their protocols are no longer standard within the expanded closed community. They will face the same difficulties that other non-standard implementations will face when trying to intercommunicate.

There are three alternatives for standardizing transport layer protocols: (i) to have one standard protocol for all types of traffic; (ii) to have a set of standard protocols based on traffic type (as defined earlier); and (iii) to allow each network to develop their own transport layer protocols, i.e., not to standardize.

When one protocol is defined to answer the needs of all users, it will probably end up not serving any very well. Its generality will require a large amount of overhead, resulting in potential severe inefficiencies. It will be extremely large, possible eliminating smaller hosts from even implementing it. This approach is not a realistic alternative.

When protocols are based on the type of traffic, one protocol per type, then each protocol can be optimized to handle the communication characteristics of the traffic for which it was intended. This alternative eliminates the need for severe overhead and size. Of course protocols will continue to be enhanced, but as long as one maintains backward compatibility, this should not present a significant problem. A host will also not have to worry about implementing many different protocols for each new communication requirement which comes along.

It should be recognized that a standard set of transport level protocols still allow for divergent hardware technologies at the network layer and minimizes the impact when a network decides to change its network technology.

Development costs would be small (except for the first round), since the same protocols developed for different machines would be available off the shelf for machines of the same type that connect to a network later.

This alternative also buffers the transport layer protocols from gateway malfunctions. If a gateway were to crash (assuming the internet control procedures are in an internet layer protocol), the transport protocol does not have to worry about messy connection cleanup, since there is no transport protocol translation at the gateway.

The third rlternative, the no standard approach, has the advantage of allowing the transport level protocols to be very finely tuned to specific applications. It does not have to compromise its technical approach for other requirements. Also, the very hard to get political agreements are not necessary for this approach.

Standardization, Some Concluding Remarks

We are in basic agreement with the studies which advocate a single standard internet layer within a community. We also contend that a set of transport layer protocols is what is required, not just a reliable data protocol. There, unfortunately, are no hard answers yet as to which way is best, because the implementations for internetworking are still in the study and experimental phases. The model we have presented in the first part of this paper is consistent with the ARPA approach to internetworking.

Should DoD choose to implement standard protocols within the context of a closed community, it is important to define that community judiciously. There are definitely impacts on DoD agencies and departments from the way other members of the same community design and implement their protocols.

The area of interconnection of computer networks is an exciting and interesting one, but many difficult questions remain unanswered.

ACKNOWLEDGMENT

The author wishes to recognize the contributions of the many individuals involved with the ARPA Internetworking Project and within the various agencies of the Department of Defense, whose conversations with the author made this paper possible. Unfortunately, they are too numerous to mention here by name.

REFERENCES

(1) Webster's New World Dictionary of the American Language, 2nd ed. (New York, 1972).

(2) Feinler, E. and Postel, J. "ARPAnet Protocol Handbook", SRI International, ARPA Netwo:k Information Center, (Menlo Park, CA, Jan 1978).

(3) Bolt, Beranek and Newman, "Specifications for the Interconnection of a Host and an IMP", Report No. 1822, (Cambridge, MA, May 1978).

(4) Cohen, D., "On Names, Addresses and Routings", ARPA Internet Experiment Note #23, Information Sciences Institute, (Marina Del Rey, CA, Jan 1978).

(5) Shoch, J., "Inter-Network Naming, Addressing, and Routing", ARPA Internet Experiment Note #19, XEROX Palo Alto Research Center, (Palo Alto, CA, Jan 1978). (Also published in COMPCON 78).

(6) Cerf, V. and Kahn, R., "A Protocol for Packet Network Intercommunication", IEEE Transactions

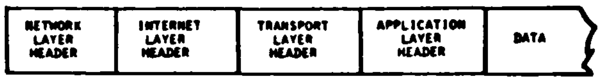
on Communications, Vol. COM-22, No. 5, (May 1974).

(7) Garlick, L., et. al. "Issues in Reliable Host-to-Host Protocols", Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, (Berkeley, CA, May 1977).

(8) Information Sciences Institute, "Internet Datagram Protocol, Version 4", ARPA Internet Experiment Note #80, (Marina Del Rey, CA, Feb 1979).

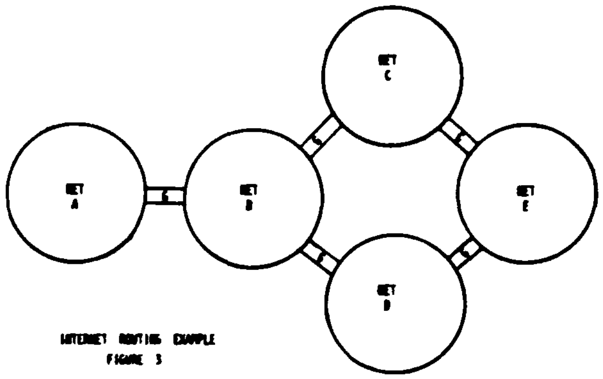
(9) Information Sciences Institute, "Transmission Control Protocol, Version 4", ARPA Internet Experiment Note #81, (Marina Del Rey, CA, Feb 1979).

(10) Sunshine, C., "Interconnection of Computer Networks", Computer Networks, Vol. 1, (1977).

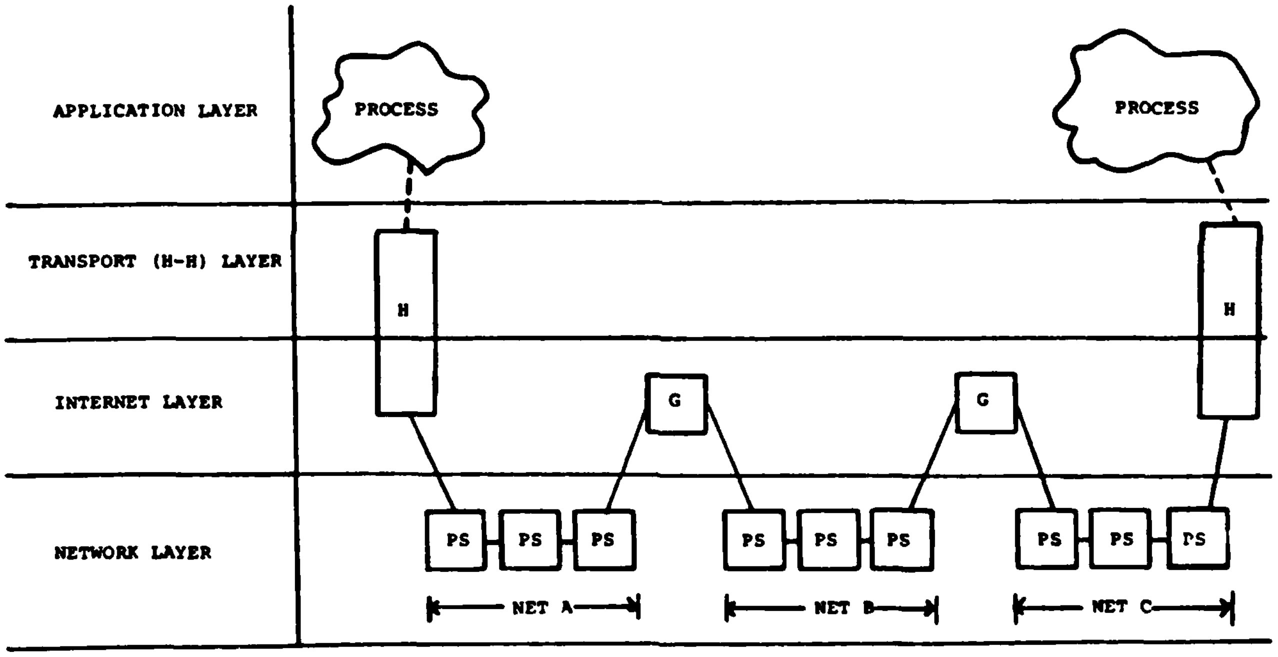


PROTOCOL LAYER HEADERS

FIGURE 2



INTERNET ROUTING EXAMPLE
FIGURE 3



H = HOST

G = GATEWAY BETWEEN 2 NETWORKS

PS = PACKET SWITCH

PROTOCOL LAYERING DIAGRAM

FIGURE 1

Internetwork Protocol Approaches

JONATHAN B. POSTEL

(Invited Paper)

Abstract—**The motivation for interconnecting networks is to provide one or more consistent services to the set of users of the interconnected networks. To provide these services either new end-to-end service protocols must be defined or the service protocols of the individual networks must be made to interwork. In either case the issues of addressing, routing, buffering, flow control, error control, and security must be considered. Two examples of interconnection strategy are examined: the interconnection of X.25 networks, and the interconnection of ARPA research networks. The models for interconnection of networks and the role of internetwork protocols are discussed.**

INTRODUCTION

THE motivations for constructing computer communication networks—data and program exchange and sharing, remote access to resources, etc.—are also motivations for interconnecting networks. This follows from the observation that the power of a communication system is related to the number of potential participants.

This paper first discusses a few key concepts involved in computer communication networks. The view that computer networks provide an interprocess communication facility is presented. The datagram and virtual circuit services are compared. The interconnection device or gateway is discussed. The relation of the interconnection issues to the open systems architecture is described.

In this paper, two approaches to internetworking are characterized: the public data network system as implied by the CCITT X.75 Recommendation and the ARPA experimental internetwork. These two systems illustrate the virtual circuit and the datagram approaches to network interconnection, respectively. The vast majority of the work on interconnecting networks falls into one of these two approaches.

INTERPROCESS COMMUNICATION

While discussing computer communication, it is useful to recall that the communication takes place at the request and agreement of processes, i.e., computer programs in execution. Processes are the actors in the computer communication environment; processes are the senders and receivers of data. Processes operate in host computers or hosts.

The protocols used in constructing the communications capability provide an interprocess communication system. Fig. 1 shows how the combination of the network and the host network interface (hardware and software) can be viewed as providing an interprocess communication system.

When a new host computer is to be connected to an existing network, it must implement the protocol layers necessary to match the existing protocol used in the network. The new host must join the network-wide interprocess communication system so the processes in that host can communicate with processes in other hosts in the network.

The interconnection of networks requires that the processes in the hosts of the interconnected networks have a common interprocess communication system. This may be achieved by converting the networks to a new interprocess communication system, by converting one or more levels of protocol to new protocols, or by translating between pairs of interprocess communication systems at their points of contact.

DATAGRAMS AND CIRCUITS

Two types of service are commonly discussed as appropriate for the network-provided interprocess communication service: datagrams and virtual circuits.

Datagrams are one-shot simple messages. They are inherently unreliable since they travel one-way and are not acknowledged. Datagrams may also arrive in a different order than sent (at least in some networks). Datagrams are simple to implement since they do not require the networks or gateways to record and update state information. Datagrams must carry complete address information in each message. The transmission of datagrams by a process is via send and receive actions.

Virtual circuits (or connections) are designed to be reliable and to deliver data in the order sent. Implementation of virtual circuits is complicated by the need for the networks or gateways to record and update state information. Virtual circuits are created through an exchange of messages to set up the circuit; when use terminates, an exchange of messages tears down the circuit. During the data transmission phase, a short form address or circuit identifier may be used in place of the actual address. To use a virtual circuit a process must perform actions to cause the virtual circuit to be created (call setup) and terminated, as well as the actions to send and receive data.

Datagrams provide a transaction type service while virtual circuits provide a connection type service. Each of these services is needed in a general purpose communication environment. Datagrams are most efficient for transaction type information requests such as directory assistance or weather reports. Virtual circuits are useful for terminal access to interactive computer systems for file transfer between computers.

Manuscript received June 15, 1979; revised December 18, 1979. This work was supported by the Advanced Research Projects Agency under Contract DAHC15 72 C 0308, ARPA Order 2223.

The author is with the Information Sciences Institute, University of Southern California, Marina del Rey, CA 90291.

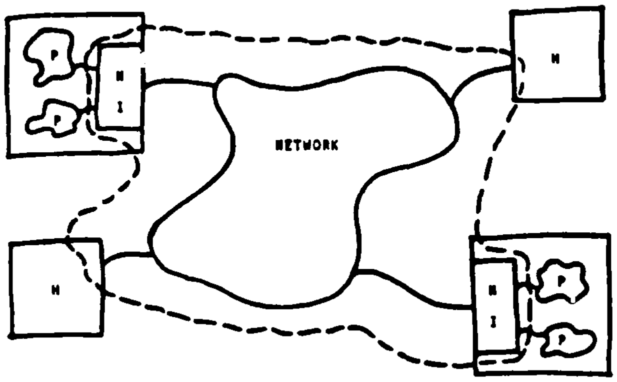


Fig. 1. Communications network.

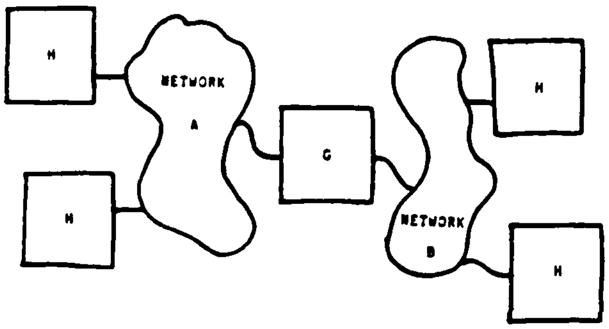


Fig. 2. Interconnected networks.

GATEWAYS

Two or more networks are connected via a device (or pair of devices) called a gateway. Such a device may appear to each network as simply a host on that network (Fig. 2).

Some gateways simply read messages from one network (unwrapping them from that network's packaging), compute a routing function, and send messages into another network (wrapping them in that network's packaging). Since the networks involved may be implemented using different media, such as leased lines or radio transmission, this type of gateway is called a media-conversion gateway.

Other gateways may translate the protocol used in one network to that used in another network by replacing messages received from one network with different messages with the same protocol semantics sent into another network. This type of gateway is called a protocol-translation gateway.

It should be clear that the distinction between media-conversion and protocol-translation is one of degree: the media-conversion gateways bridge the gap between differing link and physical level protocols, while protocol-translation gateways bridge the gap between differing network and higher level protocols.

The translation approach to network interconnection raises several issues. Success in protocol translation seems inversely correlated with the protocol level. At the lower levels, protocol translation causes no problems because the physical level and link levels are hop-by-hop in nature. It should be noted, though, that different protocols even at these low levels may have impact on the reliability, throughput, and delay characteristics of the total communication system.

At the network and transport levels, the issues of message size, addressing, and flow control become critical. Unless one requires that only messages that can be transmitted on the network with the smallest maximum message size be sent, one must provide for the fragmentation and reassembly of messages. That is, the division of a long message into parts for transmission through a small message size network, and the reconstruction of those parts into the original message at the destination. The translation of addresses is a difficult problem when one network or transport level protocol provides a larger address space than the corresponding protocol to be translated to. When end-to-end flow control mechanisms are used, as they commonly are in transport level protocols, difficulties arise when the units controlled are different. For example, when one protocol controls octets and the corresponding protocol controls letters. More difficulties arise with potential difference in the model of flow control. For example, a difference between pre- and postallocation, or between the allocation of buffer space and the allocation of transmission rate.

At higher levels, the problems are more difficult because of the increased state information kept and the lower likelihood of one-to-one translation of individual protocol messages. A further difficulty is that each level further multiplexes the communication so that each connection or stream or channel or virtual circuit must be separately translated. It should be noted that neither of the specific interconnection approaches discussed in this paper attempts higher level protocol translation.

Gateways may be thought of as having a "half" for each network they interconnect. One could model the operation of a gateway as having each gateway-half contain procedures to convert from a network specific protocol into a standard protocol and vice versa (Fig. 3).

RELATION TO OPEN SYSTEMS ARCHITECTURE

In relation to the open systems architecture, the interconnection of networks focuses on levels 3 and 4 [1].

To review, the open systems architecture defines the following levels of protocol:

Level	Function
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Link
1	Physical

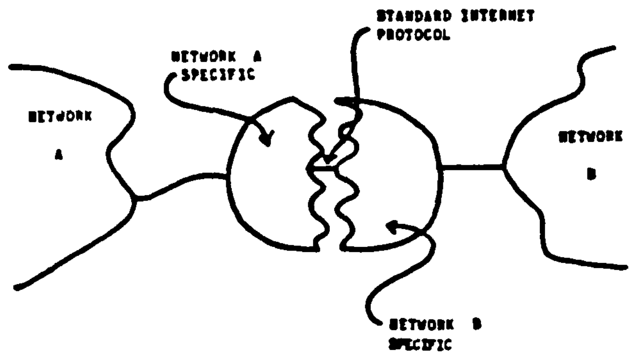


Fig. 3. Gateway halves.

The lower levels, the physical and the link levels, are hop-by-hop in nature and present no interconnection issues in terms of compatibility, although there may be some performance concerns.

The higher levels, the session level, the presentation level, and the application level, have so many compatibility requirements that it seems quite unlikely that interconnection of different protocols at those levels will be workable.

Thus, it is at the network level and the transport level that the interconnection of networks finds issues of concern.

The network level corresponds to the interface to datagram service, and the transport level corresponds to the interface to virtual circuit service.

In some networks, the network level and datagram service have been hidden from the user, forcing consideration of network interconnection at the transport level.

INTERCONNECTION OF X.25 NETWORKS

Introduction

The public data networks (PDN's) that follow the CCITT X.25 Recommendation [2] are to be interconnected via an interface specified in CCITT Recommendation X.75 [3]. Recommendation X.25 specifies the interface between the customer's equipment, called the data terminal equipment (DTE); and the network equipment, called the data circuit-terminating equipment (DCE). Recommendation X.25 implies a virtual circuit operation. Thus, the PDN's offer an interface to a virtual circuit transport level protocol. Fig. 4 shows the model of a PDN virtual circuit.

The interface between two PDN's specified in Recommendation X.75 is quite similar to that in Recommendation X.25. The equipment on either side of this interface is called a signaling terminal (STE). The STE-STE interface is much like the DTE-DCE interface. The STE-STE interconnection is a split gateway with each gateway-half in a physical device controlled by the PDN connected to that gateway-half. Fig. 5 shows the interconnection of PDN's.

The interconnection of PDN's via X.75 interfaces results in a series of virtual circuits. Each section is a distinct entity with separate flow control, error recovery, etc. Fig. 6 shows a PDN transmission path with two virtual circuits (VC's) and five separate flow control (FC) steps.

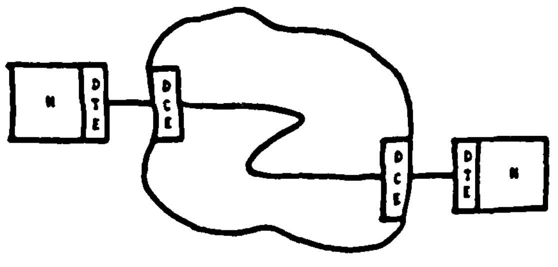


Fig. 4. PDN virtual circuit.

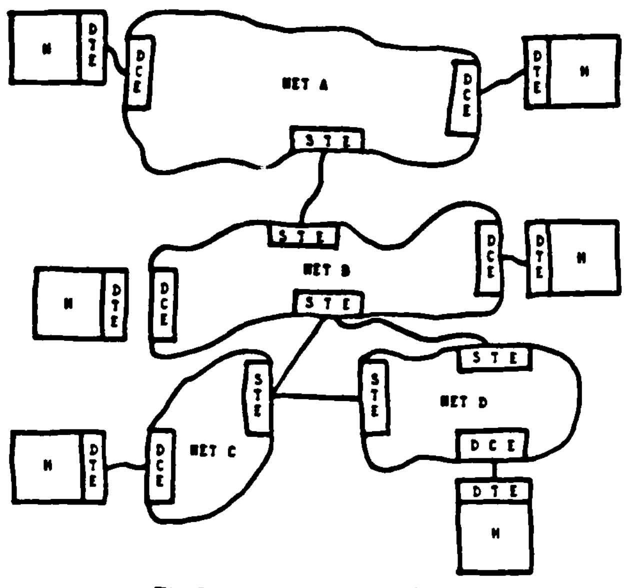


Fig. 5. Interconnection of PDN's.

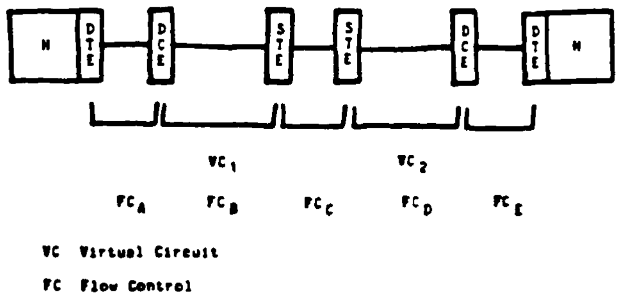


Fig. 6. PDN transmission path.

Addressing

The address field is variable in length up to 15 digits, with each digit coded in a 4 bit field. The maximum address is then 60 bits (about 8 octets).

Routing

The user has no influence over routing used. To create the series of virtual circuits, a series of call setups establishes a fixed route (between pairs of STE's at least). State information must be kept for each call in the source and destination DTE's and DCE's and in each STE in the route.

Buffering and Flow Control

Each portion of the total path is a distinct virtual circuit. Each virtual circuit has an independent flow control (and particular to that PDN). In addition, there is flow control across each STE-STE interface. All this flow control is on a per call basis. This stepwise flow control may introduce delay in the total path that could be avoided with an end-to-end scheme.

There are some concerns about the interaction of two types of flow control implemented in PDN's. One type allows one message in transit from source DCE to destination DCE at any one time. The other allows multiple messages to be in transit, the number being determined by the flow control window.

Acknowledgment

Each portion of the total path has an acknowledgment. The user to network interface also has an acknowledgment. This local acknowledgment means only that the first PDN has accepted the message for transmission, not that it has arrived at the destination.

Recovery

The X.25 and X.75 Recommendations do not specify how the PDN's deal with errors internally. If unrecoverable errors occur, the network will signal a Reset, which apparently means that the virtual circuit still exists, but the flow control is reset and messages may have been lost. More serious errors result in the call being cleared.

Because of the fixed route nature of the multinetwork path, an STE failure disrupts the communication.

Security

The X.25/X.75 Recommendations do not provide any security features.

Header Structure

Once the call is established, a header is only 3 octets. The call setup headers are substantially longer, typically 20 octets, but possibly as large as 166 octets. There is a tradeoff between header size and state information kept; in the PDN's, the tradeoff has been made toward small headers and large state. The details of the headers are shown in Appendix I.

Summary

The most important aspect of the interconnection of PDN's is that service provided to the using process is a virtual circuit with essentially the same properties a single PDN would have provided. This is done by concatenating a series of virtual circuits to provide the total path, resulting in a fixed route through a set of network interconnection points.

INTERCONNECTION OF ARPA RESEARCH NETWORKS

Introduction

The ARPA sponsored research on interconnections of networks has let to a two-level protocol to support the equivalent function of the PDN's X.25/X.75 service. The ARPA sponsored work on networks has developed an internet protocol (IP) [4], and a transmission control protocol (TCP) [5].

TCP is a logical connection transport protocol and is a level 4 protocol in the OSA model of protocol structure.

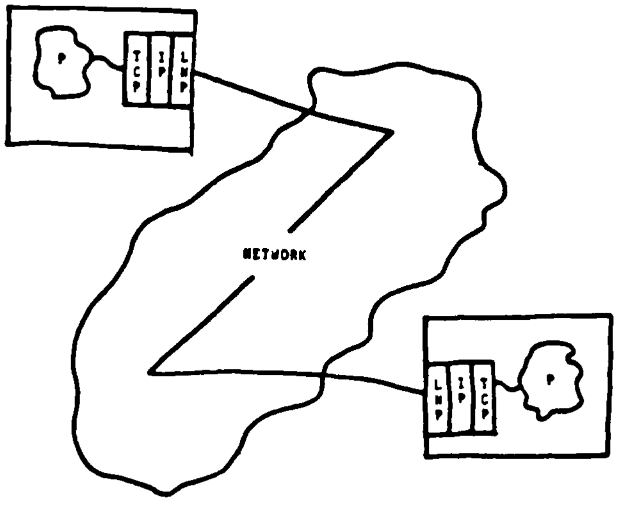


Fig. 7. End-to-end connection.

The IP is a datagram protocol. The collection of interconnected networks is called an internet. IP is the network protocol of the internet and this is a level 3 protocol in the OSA model. The actual networks used are of various kinds (e.g., the ARPANET, radio networks, satellite networks, and ring or cable networks) and are referred to as local networks even though they may span continents or oceans. The interface to a local network is a local network protocol or LNP. Fig. 7 shows the model of an end-to-end connection.

In the ARPA model, the networks interconnect via a single device called a gateway. A gateway is a host on two or more networks. Fig. 8 shows the ARPA model of the interconnection of networks.

Each network addresses a gateway on it in the same way it addresses any other host on it. The information required to deliver a message to a destination in the internet is carried in the IP header. The IP is implemented in the gateways and in hosts. A sending host prepares a datagram (which is an IP header and the original message) and then selects a gateway in its own net to forward the datagram. The sending host then sends the datagram wrapped in a local network packet to that gateway.

A gateway receives a packet from one of the local networks to which it is attached, and unwraps the IP datagram. The gateway then examines the IP header and determines the next gateway (or destination host) address in one of the local networks it is directly connected to. The gateway then sends the datagram with its IP header in a new local net packet to that gateway (or host).

The IP has no provision for flow control or error control on the data portion of the message (the IP headers are checksummed). There are no acknowledgments of IP messages. The IP is simple and the gateway may be implemented in small machines. A key point is that a gateway has no state information to record about a message. At the IP level, there are no connections or virtual circuits.

The IP does not provide a service equivalent to the PDN's X.25/X.75. To provide that type of end-to-end reliable ordered delivery of data the ARPA internet uses TCP.

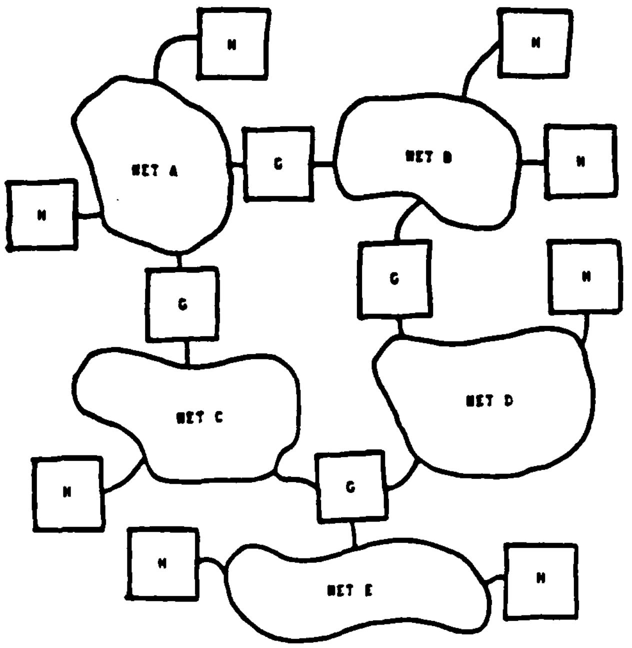


Fig. 8. ARPA model of interconnection of networks.

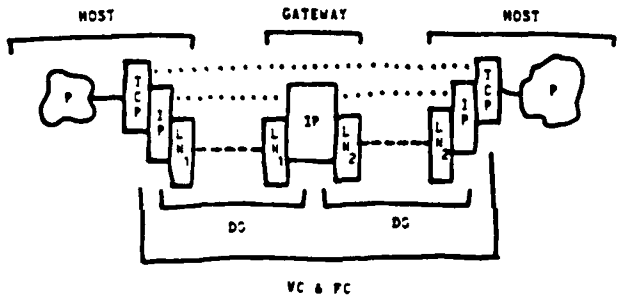


Fig. 9. ARPA model of transmission path.

TCP uses end-to-end mechanisms to ensure reliable ordered delivery of data over a logical connection. It uses flow control, positive acknowledgments with time out and retransmission, sequence numbers, etc., to achieve these goals. Fig. 9 shows the conceptual transmission path in this interprocess communication system, pointing out the datagram (DG) path between the IP modules and the virtual circuit path between the TCP modules at the source and destination and the flow control (FC) at that level.

ARPA has used these techniques to interconnect several very different networks including the ARPANET, packet radio nets, a satellite net, and several local networks.

Addressing

The size of the address in this experimental system is fixed. The IP provides a one octet network field and a three octet host field. Also a one octet protocol identifier in the IP header may be considered address information. The TCP provides a two octet port field. The total of the address length is then seven octets. Provision has been made for a host to have several addresses, so the host field is sometimes called the logical host field. The total address is the concatenation of the network, host, protocol, and port fields.

Routing

Normally, the user has no influence over the route used between the gateways. There is no call setup and the route may vary from one message to the next. No state information is kept in the gateways.

A user might insert a source routing option in the IP header to cause that particular message to be routed through specific gateways.

Buffering and Flow Control

There is no flow control mechanism in the IP. The gateways do not control the flow on connections for they are unaware of connections or any relation between one message and the next message. The gateways may protect themselves against congestion by dropping messages. When a gateway drops a message because of congestion, it may report this fact to the source of the message.

The TCP uses end-to-end flow control using windows on a per logical connection basis.

Acknowledgment

The IP has no provision for acknowledgments. The TCP uses acknowledgments for both error control and flow control. The TCP acknowledgments are not directly available to the user.

Recovery

Errors in a network or gateway result in a message being dropped, and the sender may or may not be notified. This inherent unreliability in the IP level allows it to be simple and requires the end-to-end use of a reliable protocol.

TCP provides the reliable end-to-end functions to recover from any lost messages. The TCP uses a positive acknowledgment, time out, and retransmission scheme to ensure delivery of all data. Each message is covered by an end-to-end checksum.

Because of the potential of alternate routing, the end-to-end communication may be able to continue despite the failure of a gateway.

Security

The IP provides an option to carry the security, precedence, and user group information compatible with AUTODIN II. The enforcement of these parameters is up to each network, and only AUTODIN II is prepared to do so.

The TCP end-to-end checksum covers all the address information (source and destination network, host, protocol, and port), so if the checksum test is successful the address fields have not been corrupted.

Header Structure

The IP header is 20 octets (plus options, if used), but there is no call setup and no gateway state information. Thus, at the IP level, the header size versus state information tradeoff has been made toward large header and little (no) state information.

The TCP header is 20 octets (plus option, if used). There is a connection establishment procedure called the "three-way handshake," and significant state information is kept. In this case, there are both large headers and large state tables. The details of the headers are shown in Appendix II.

Summary

The ARPA networks are interconnected by using a common datagram protocol to provide addressing (and thus routing) information and an end-to-end transport protocol to provide reliable sequenced data connections.

This model has evolved from the ARPANET experience, in particular from the internetwork protocol model suggested in a paper by Cerf and Kahn [6].

CONCLUSION

Both the PDN's and the ARPA networks are interconnected by establishing standard protocols. The PDN's provide a virtual circuit service by concatenating the virtual circuit services of the individual networks. The ARPA networks use two levels of protocol to provide both datagram and virtual circuit services.

Additional discussion of the interconnection of PDN's is provided in [7], [8]. In another paper in this issue Boggs *et al.* present in detail another example of network interconnection using the datagram approach [9].

The issues of network interconnection have been discussed for at least 5 years (for example, McKenzie [10]). The recent expositions by Sunshine [11], Cerf and Kirstein [12], and Gien and Zimmermann [13], are particularly recommended.

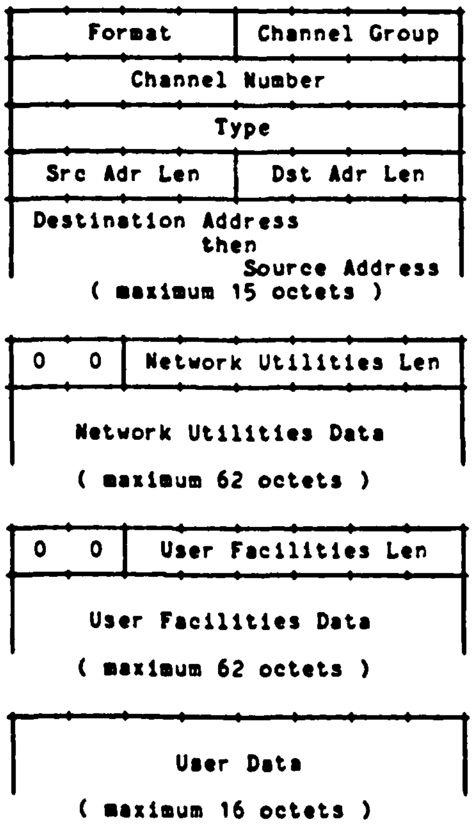
APPENDIX I

X.75 HEADER FORMATS

The call request and the data packet formats are illustrated here. These typify the X.75 packet formats. All the X.75 packets are the same in the first two octets. The format field indicated the type of packet.

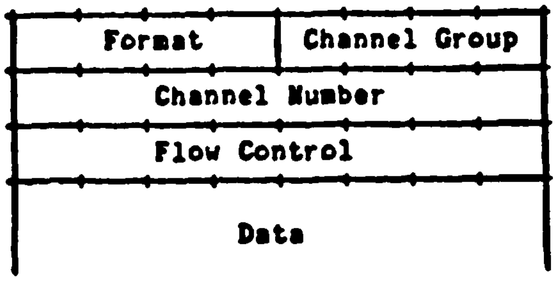
Call Request

The call request packet is variable in length from a practical minimum of 11 octets to an unlikely maximum of 160 octets.



Data

The Data packet has a three octet header.



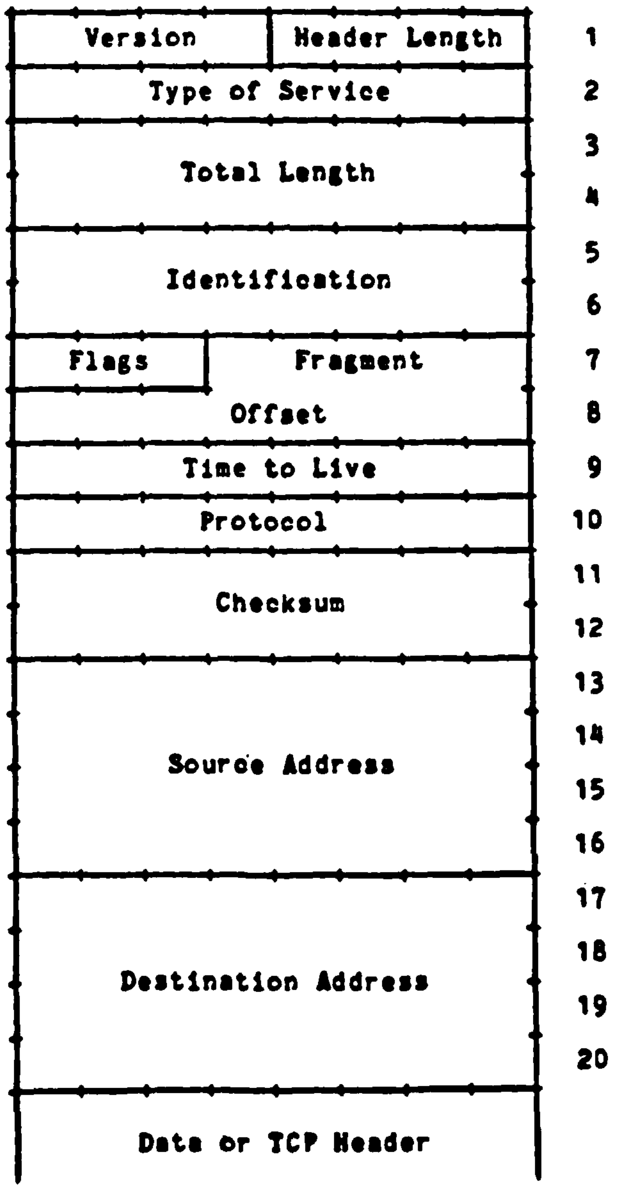
APPENDIX II

ARPA PROTOCOL HEADER FORMATS

Every datagram carries the basic IP header. Every TCP segment transmitted carries the basic TCP header.

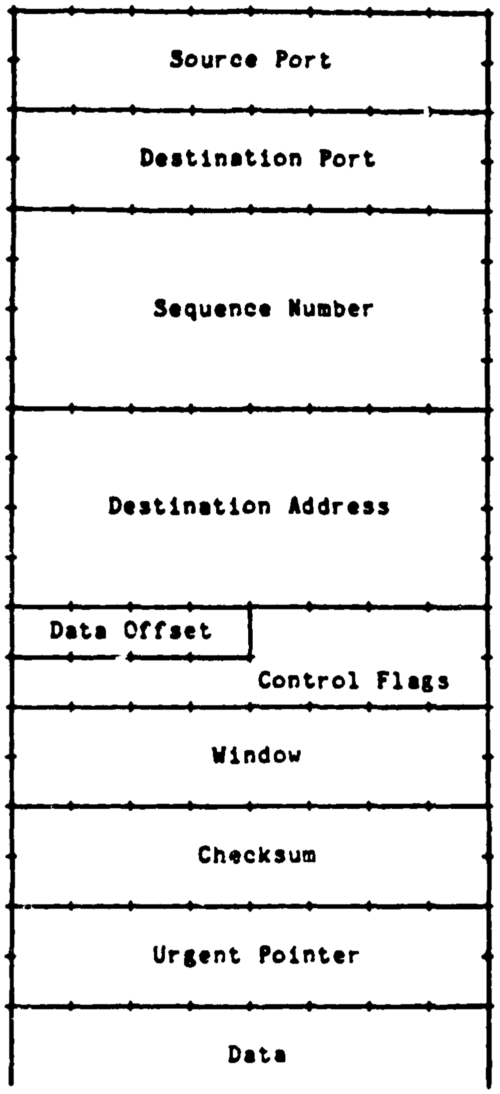
Internet Protocol

The ARPA IP has a basic header of 20 octets, and may carry a variable number of options up to a total length of 60 octets.



Transmission Control Protocol

The basic TCP header is 20 octets, and the header may be up to 60 octets long if options are used.



REFERENCES

[1] H Zimmermann. "The ISO reference model," this issue, pp 425–432
[2] "Recommendation X 25/Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) for terminals operating in the packet mode on public data networks," in *CCITT Orange Book*, vol 7, Int Telephone and Telegraph Consultative Committee, Geneva, Switzerland
[3] "Proposal for provisional Recommendation X 75 on international interworking between packet switched data networks," in CCITT Study Group VII Contribution no 207, Int Telephone and Telegraph Consultative Committee, Geneva, Switzerland, May 1978
[4] DARPA, "DOD standard internet protocol," IEN-128, Defense Advanced Research Projects Agency, Jan 1980
[5] DARPA, "DOD standard transmission control protocol," IEN-129, Defense Advanced Research Projects Agency, Jan 1980.
[6] V Cerf and R Kahn, "A protocol for packet network intercommunication," *IEEE Trans Commun*, vol COM-22, pp 637–648, May 1974
[7] G Grossman, A Hinchley, and C Sunshine, "Issues in international public data networking," *Comput Networks*, vol 3, pp 259–266, Sept. 1979.
[8] V DiCiccio, C Sunshine, J Field, and E Manning, "Alternatives for interconnection of public packet switching data networks," in *Proc Sixth Data Commun Symp*, ACM/IEEE, Nov. 1979, pp. 120–125.
[9] D. Boggs, J. Shoch, E Taft, and R Metcalfe, "Pup An internetwork architecture," this issue, pp 612–624.
[10] A. McKenzie, "Some computer network interconnection issues," in *Proc. Nat. Comput Conf*, AFIPS, 1974, pp 857–859

★

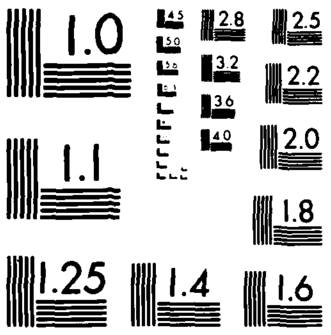
Jonathan B. Postel received the B S and M S degrees in engineering and the Ph D degree in computer science from the University of California, Los Angeles.

He has worked for the MITRE Corporation in McLean, VA, and SRI International in Menlo Park, CA. At UCLA he was involved in the development of the ARPANET Network Measurement Center and the installation of the first host on the ARPANET Since that time, he has participated in the development of many of the higher level protocols used in the ARPANET. He is currently a Computer Scientist at the USC/Information Sciences Institute in Marina del Rey, CA, where his research focuses on the interconnection of computer networks.

AD-A153 624 INTERNET PROTOCOL IMPLEMENTATION GUIDE(U) SRI INTERNATIONAL MENLO PARK CA NETWORK INFORMATION CENTER AUG 82 DCA200-83-C-0025 2/2

UNCLASSIFIED F/G 17/2 NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

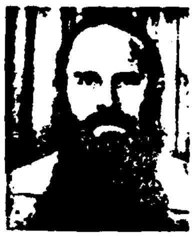
The ARPA Internet Protocol

Jonathan B. Postel, Carl A. Sunshine and Danny Cohen

Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, USA

A variety of computer networks are interconnected by gateway computers in the ARPA internetwork system. Processes on different networks may exchange messages with each other by means of an Internet Protocol which must be implemented in each subscriber (host) computer and in the gateways. The Internet Protocol is a relatively simple protocol that provides for the delivery of individual messages (datagrams) with high but not perfect reliability. This Internet Protocol does not replace the existing protocol in any network, but is used by processes to extend the range of communications. Messages in Internet Protocol are transmitted through any individual network by encapsulating them in that network's protocol. This paper presents an overview of the Internet Protocol and the operation of the gateway computers in the ARPA internet system.

Keywords: Protocol, ARPA Net, Internetwork, Datagram, Gateway.



Jonathan Postel received his B.S. and M.S. degrees in Engineering and his Ph.D. in Computer Science from the University of California, Los Angeles. Dr. Postel has worked for the MITRE Corporation in McLean, Virginia and SRI International in Menlo Park, California. He is currently a Computer Scientist at the University of Southern California Information Sciences Institute in Marina del Rey, California. At UCLA he was involved in the development of the ARPANET Network Measurement Center and the installation of the first host on the ARPANET. Since that time, he has participated in the development of many of the higher level protocols used in the ARPANET. His current research focuses on the interconnection of computer networks.

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 72C 0308, Arpa Order No. 2223. The views and conclusions in this study are the authors' and should not be interpreted as representing the official opinion or policy of ARPA, the U.S. Government or any other person or agency connected with them.

1. Introduction

The family of computer networks developed for the United States Defense Advanced Research Projects Agency (DARPA) represents one of the largest and most diverse internetwork systems currently in operation. The basic approach to interconnecting this variety of networks was developed over several years, and has resulted in the definition of an Internet Protocol (IP) [1]. This paper is intended primarily to document the details of the IP in the open literature, and secondarily to provide a brief discussion of the major design tradeoffs which caused the IP to take its current form.

Section 2 presents an overview of the DARPA approach to interconnection and the operation of IP. Section 3 details IP's main features, while some additional options are treated in section 4. Section 5 summarizes the IP and other functions performed in the *gateways* which interconnect networks. Section 6 discusses the major design choices in developing IP. Section 7 outlines several questions and extensions requiring further work.

Carl Sunshine is with the University of Southern California Information Sciences Institute where he is engaged in research on computer networks and their protocols. He is particularly interested in protocol modeling, and analysis, and network interconnection. Sunshine received a PhD in computer science from Stanford University in 1975, and worked at the Rand Corporation from 1975 to 1979. He is active in IFIP TC6.1 (the Internetwork Working Group) and ISO SC16, is Secretary/Treasurer of ACM SIGCOMM, and serves on the editorial board of Computer Networks journal.

Dan Cohen was born in Haifa, Israel, in 1937. He received the B.Sc. degree in mathematics from the Technion-Israel Institute of Technology in 1963, and the Ph.D. degree in computer science from Harvard University, Cambridge, MA, in 1969.

Since 1969, he has been on the faculty of Harvard University, Technion-Israel Institute of Technology, and the California Institute of Technology, Pasadena. In 1973, he joined the Information Sciences Institute of the University of Southern California, Los Angeles, here he leads several computer network-related projects. His research interests include interactive realtime systems, graphics voice communication, networking and computer architecture.

2. Overview

Since the development of the ARPANET in the early 1970's, a variety of new packet switching network technologies and operational networks have been developed under DARPA sponsorship, including satellite, packet radio, and local networks. In order to allow processes on different networks to communicate with each other, a means for interconnecting networks has been developed without requiring changes to the internal operation of any network.

The method chosen for interconnecting networks makes minimal demands on individual networks. To facilitate inclusion of a wide variety of networks, each net is required to provide only a minimal *datagram* level of service (i.e. to deliver individual packets of moderate length between its users with high but not perfect reliability). Networks are inter-connected by gateway computers that appear to be local subscribers on two or more nets. The gateways are responsible for routing traffic across multiple networks, and for forwarding messages across each net using the packet transmission protocol in each network. The gateways provide a point-to-point internet datagram service by concatenating the datagram services available on each individual net. Such a system of interconnected networks has been called a *Catenet* [2].

This approach allows the interconnection of networks that have significantly different internal protocols and performance. The networks in the ARPA-Catenet were originally designed as independent entities. In the Catenet approach no changes are required in the internal functions of any network.

Gateways provide an internet service by means of an Internet Protocol (IP) that defines the format of internet packets and the rules for performing internet protocol functions based on the control information (internet header) in these packets. IP must be implemented in host computers (subscribers) engaged in internet communication as well as in the gateways. Gateways also use a gateway-to-gateway protocol to exchange routing and control information.

IP provides for transmitting datagrams from an internet source to an internet destination, potentially in another net. IP also provides for *fragmentation* and *reassembly* of long datagrams, if necessary, for transmission through networks with small packet size limits.

IP is purposely limited in scope to provide only the function necessary to deliver datagrams over an interconnected system of networks. The functions of flow control, sequencing, additional data reliability, or other services commonly found in host-to-host protocols, and multidestination delivery capability or other services are purposely left for higher level protocols to provide as necessary. This allows the higher levels to be tailored to specific applications, and allows a simple and efficient implementation of IP.

2.1. Place in Protocol Hierarchy

As described above, IP functions on top of, or uses, the packet transmission protocol in each individual network. IP is used by higher level end-to-end protocols such as a reliable transport protocol, e.g., Transmission Control Protocol (TCP) [3] in the ARPA-Catenet or a "real time" protocol, e.g., for packet speech.

As shown in Figure 1, IP is the only level in the protocol hierarchy where a single common protocol is used. By locating this point of convergence at the internet datagram level, the Catenet approach preserves the flexibility to incorporate a variety of individual networks and protocols providing packet transmission below IP, while remaining general and efficient enough to serve as a common basis for a variety of higher level protocols. With this approach,

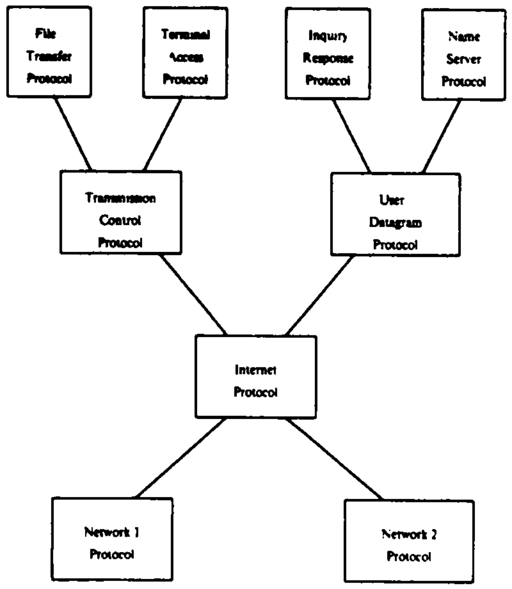


Fig. 1. Protocol Hierarchy.

gateways need only provide datagram service, and remain relatively simple, inexpensive, and efficient.

2.2. Model of Operation

The Internet Protocol provides two major functions: routing a datagram across successive networks to its internet destination address, and fragmentation/reassembly of large packets when needed to cross nets with small packet size limits. To accomplish this, an IP module must reside in each host engaged in internet communication and in each gateway that interconnects networks. The following scenario describes the progress of a datagram from source to destination (assuming one intermediate gateway is involved-see Figure 2).

The basic notion is *encapsulation*. The data to be transmitted must pass through a variety of network environments. To do this the data is encapsulated in an internet datagram. to send the datagram through an individual network, it is in turn encapsulated in a local network packet, and extracted at the other side of that network where it is decapsulated from the first network protocol and is encapsulated in the second network protocol. Thus the model is a series of encapsulation/extractions, not translations. This encapsulation is an information preserving transformation, all the information is preserved even if the individual network cannot make use of it.

The sending internet user (typically a higher level protocol module such as TCP) prepares its data and calls on its local IP module to send the data as a datagram, passing the destination address and other parameters as arguments of the call.

The IP module encapsulates the data in a datagram and fills in the datagram header. The IP module examines the internet destination address. If it is on the same network as this host, it sends the datagram directly to the destination. If the datagram is not on the same network then the IP module sends the datagram to a gateway for forwarding. The selection of which gateway to send the datagram to is an internet routing decision.

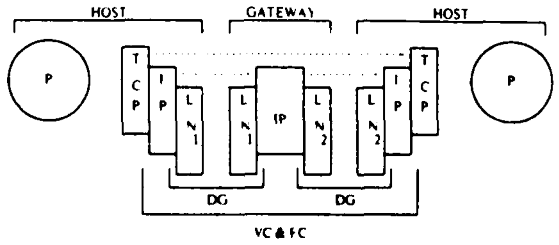


Fig. 2. ARPA Model Transmission Path.

The local network interface (note that from the IP point of view, all actual networks are "local" even if they span across the world) creates a local network packet with its own header, and encapsulates the datagram (complete with internet header) in it, then sends the result via the local network.

The datagram arrives at a gateway host encapsulated in the local network packet. The local network interface extracts the IP datagram and turns it over to the IP module.

The IP module determines from the internet destination address that the datagram should be forwarded to another host in a second network. The IP module uses the local portion of the destination address to determine the local net address for the destination host. It calls on the local network interface for the second network to send the datagram to that address.

If the datagram is too large to be sent through the second network, the IP module fragments it into several smaller datagrams and passes each one :o the local net interface.

The local network interface creates a local network packet and encapsulates the datagram, sending the result to the destination host. At the destination host, the datagram is extracted from the local net packet and passed to the IP module.

The IP module determines that the datagram is for an internet user in this host. If the datagram is a fragment, the IP module collects all fragments of a particular datagram and reassembles the complete original datagram. It then passes the data to the user along with the internet source address and other information from the internet header.

2.3. Additional Mechanisms

In addition to the basic addressing and fragmentation functions described above, IP uses four key mechanisms in providing its service: *Type of Service, Time to Live, Options*, and *Header Checksum*. Each of these is summarized here and fully described in Sections 2 and 3.

The Type of Service (TOS) is used to indicate the quality of the service desired – this may be thought of as selecting among Interactive, Bulk, or Real Time, for example. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the Catenet. This type of service information is used by gateways to select the actual parameters for transmission through each individual network.

The Time to Live (TTL) is an indication of the lifetime of a datagram. Datagrams must not be allowed to persist in the ARPA-Catenet indefinitely. This is because reliable end-to-end protocols depend on there being an upper bound on datagram lifetime, especially old duplicates due to retransmissions. The time to live can be thought of as a self-destruct time limit.

The Options provide for control functions useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, error reports, and special routing.

The Header Checksum provides a verification that the information used in processing the datagram has been transmitted correctly. However, the data is not covered by the checksum, and may contain errors (see Section 2.6). If the header checksum fails, the internet datagram is discarded by the entity which detects the error.

2.4. Relation to Other Work

The current ARPA Internet Protocol evolved from ideas suggested by Cerf and Kahn [4], and from contemporaneous proposals within the International Federation for Information Processing (IFIP) Technical Committee 6.1 (also known as the International Network Working Group or INWG), in which internet functions and reliable transport functions were combined in a single protocol. Subsequent development of other high level protocols (such as packet speech) that needed internet services led to splitting internet functions and reliable transport functions into separate protocols (the current IP and TCP).

The Internet Protocol used in the ARPA-Catenet is quite similar in philosophy to the PUP protocol [5] developed by the Xerox Corporation. The PUP protocol does not include fragmentation (leaving this to each local net to perform if necessary), but does include a third level of addressing (Ports within hosts) in the internet packet header. IP and PUP share the important principle of having a single common internet datagram protocol as a point of convergence in their protocol hierarchies. Both the PUP and IP systems use the encapsulation technique, and a scheme for "mutual encapsulation" has been worked out [6]. PUP and IP both trace their roots to a joint XEROX-DARPA project at Stanford University. The network interconnection approach used by the European Informatics Network [7] is also quite similar.

Public packet switching networks, on the other hand, have chosen to use virtual circuit (VC) level of service as the level of interconnection, providing end-to-end service as a concatenation of VCs through each network. Since gateways must participate at the VC level, they are more complex and costly, and the end-to-end service may be less efficient and less robust. They are also unable to accommodate "transaction" type users without setting up a VC, although the CCITT is currently considering adding a datagram type of service. For further comparison of CCITT and Catenet approaches see [8–12].

In summary, the ARPA Internet Protocol supports delivery of datagrams from an internet source to a single internet destination. IP treats each datagram as an independent entity unrelated to any other datagram. There are not connections or logical circuits (virtual or otherwise). There are no acknowledgements either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is minimal flow control. For flexibility, it is explicitly left to higher level protocols to provide these functions.

3. Main Features

The following paragraphs describe in some detail the mechanisms of the IP. A summary of the contents of the IP header is shown in Figure 3. Further information may be found in the current specification [1].

3.1. Addressing

The IP provides a two level addressing hierarchy. The upper level of the hierarchy is the *network number* (8 bits), and the lower level is an address within that network (24 bits), and is commonly called the *host*. This second level of the hierarchical address is sometimes called the *local address*. The details of the local address are dependent on the particular network.

The local address should allow a single physical host to act as several logically distinct internet hosts. That is, there should be mapping between internet

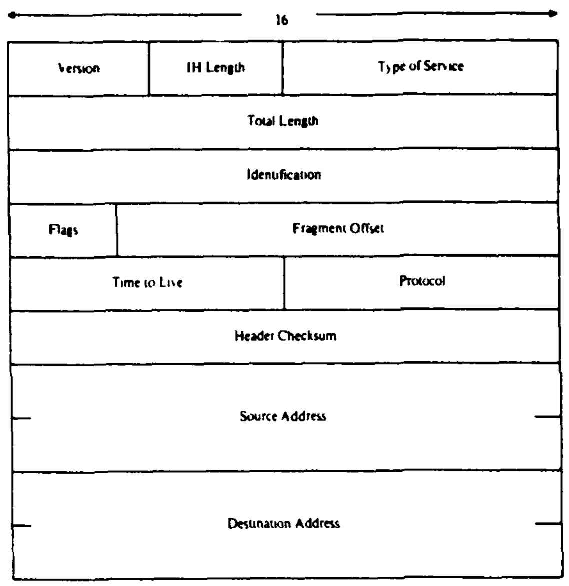


Fig. 3. INTERNET Protocol Header.

host addresses and network/host interfaces that allows several internet addresses to correspond to one physical interface. It should also be possible for several interfaces to accept or emit datagrams for the same internet address.

3.2. Protocol Number

The *Protocol Number* indicates the next level protocol used in the data portion of the datagram. This allows the internet module to demultiplex the incoming datagrams to higher level protocol modules for further processing. Hence, the protocol number indicates the format for parsing the rest of the datagram. Note that there is only one protocol number rather than a source protocol and a destination protocol because, higher level protocol modules exchange datagrams with each other using the same protocol. For example, two TCP modules exchange TCP segments via datagrams marked "TCP" in the protocol number.

One particular protocol number designates a multiplexing protocol which allows several independent data blocks from possibly different higher level protocol modules to be aggregrated together into one datagram for transmission [13].

3.3. Fragmentation and Reassembly

The IP provides information to allow datagrams to be fragmented for passage through networks with small packet size limits and to be reassembled at the destination. The necessary information includes an identification of the fragments that belong to the same datagram and the position of each fragment within the datagram.

The *Identification* (ID) field is used together with the source and destination address, and the protocol number, to identify datagram fragments to be assembled together. The *More Fragments* flag (MF) is set if the datagram is not the last fragment. The *Fragment Offset* (FO) identifies the fragment location, relative to the beginning of the original unfragmented datagram. These offsets are counted in units of 8 octets. Hence, if a datagram is fragmented, its data portion must be broken on 8 octet boundaries. This convention is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, FO = 0).

If the *Don't Fragment* flag (DF) is set, then internet fragmentation of this datagram is not permitted, although this may force it to be discarded at a gateway to a small packet network. DF can be used to prohibit fragmentation in cases where the receiving host does not wish to reassemble internet fragments. It is also possible that a small packet network could use network specific fragmentation and reassembly without the knowledge or involvement of the IP modules [14].

If a datagram is too large to be forwarded through any net, the entrance gateway breaks it into as many fragments as are necessary to fit within that net's packet size limit. Figure 4 shows a large datagram of 452 octets being fragmented into two smaller fragments (only the header fields relevant to fragmentation are given). Subsequent gateways may break the fragments into even smaller fragments if necessary using the same procedure.

Datagrams arriving at the destination IP are easily recognizable as fragments if either MF or FO is nonzero. Fragments from the same original datagram are identified by having identical ID fields (for a particular source, destination, and protocol number). Fragments are queued until the original datagram can be fully reassembled. Reassembly may be accomplished by placing the data from each fragment in a buffer at the position indicated by FO. Using the header information from the first fragment, the reas-

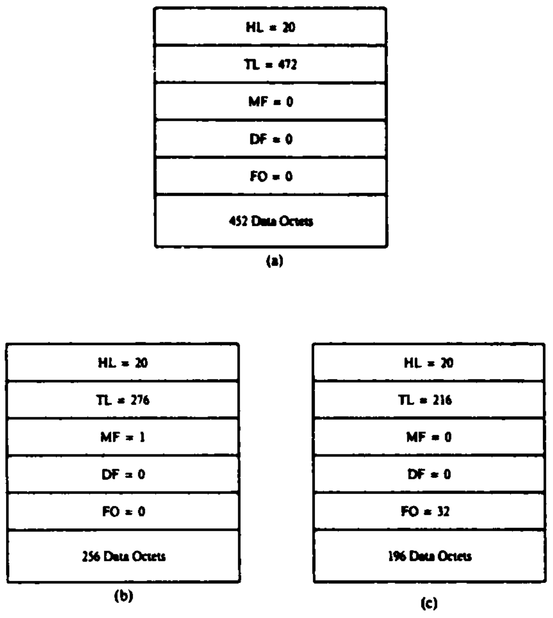


Fig. 4. Fragmentation Example.

sembled datagram is processed further just as if it had been received intact. If the time to live on any fragment expires during reassembly, the partially assembled datagram is discarded, and an error datagram is sent to the source.

A convention has been established in the current ARPA-Catenet that no datagrams larger that 576 octets will be sent, and that all receivers will be prepared to receive a reassemble datagrams up to this length (unless specifically arranged otherwise). This number is chosen to allow a data block of 512 octets and a reasonable number of header octets for several protocol levels to be transmitted in one datagram. Note that the IP header is repeated in each fragment. Hence, the minimum maximum packet size for any network in the Catenet is 20 header octets plus 8 data octets or 28 octets total.

The internet fragmentation procedure allows the fragments to be treated as independent datagrams the rest of the way to their destination (even taking different routes), with reassembly occurring only at the destination.

There is a need to uniquely identify the fragments of a particular datagram. Hence the sender must choose the identification field to be unique for each source/destination pair and protocol number for the time the datagram (or any fragment of it) could exist in the internet. Since the ID field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is beneficial for some higher level protocols to choose the identification field. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment. Note that a retransmission might be routed via a different set of networks and gateways and also may be fragmented into a different number of different sized fragments. The fragmentation information permits reassembly from fragments from either copy of the datagram.

3.4. Type of Service

The Type of Service (TOS) provides a network independent indication of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Some networks offer several precedence levels of service. Another choice involves a low-delay vs. high-reliability trade off. Typically networks invoke more complex (and delay producing) mechanisms as the need for reliability increases. A few networks offer a stream service, whereby one can achieve a "smoother" service at some cost. Typically this involves the reservation of resources within the network.

The abstract service quality parameters provided by IP are:

Precedence: Indicates the importance of this datagram.

Stream or Datagram: Indicates if there will be other datagrams from this source to this destination at regular frequent intervals justifying the maintenance of stream processing information.

Reliability: A measure of the level of effort desired to ensure delivery of this datagram.

Speed: A measure of the importance of prompt delivery of this datagram.

Speed over Reliability: Indicates the relative importance of speed and reliability when a conflict arises in achieving both.

3.5. Time to Live

The Time to Live (TTL) indicates the maximum time the datagram is allowed to exist in the Catenet.

As a datagram moves through the Catenet the TTL is decremented. If the TTL reaches zero the datagram should be discarded. The intention is to cause long delayed or undeliverable datagrams to be discarded. Guaranteeing a maximum lifetime for datagrams is important for the correct functioning of some higher level protocols such as TCP, and to protect the Catenet resources.

This field should be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no information is available on the time actually spent, the field should be decremented by 1. The time is measured in units of seconds, and the maximum TTL is 255 seconds.

3.6. Checksum

The IP provides a checksum on the header only. Since some header fields may change (e.g., TTL, MF, FO), this is recomputed and verified at each point that the internet header is processed. This is a hop-by-hop checksum.

This checksum at the internet level is intended to protect the internet header fields from transmission errors. If the internet header contained undetected errors, misrouting and other unanticipated behavior could result. There may be applications in which it is desirable to receive data even though there are a few bit errors. If the IP enforced a data checksum and discarded datagrams with data checksum failures such applications would be restricted unnecessarily.

The checksum is computed as the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero. This checksum is simple to compute and has been adequately reliable for usage to date, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

3.7. Header Format

In addition to the main features discussed above, the IP includes the following items in the datagram header:

A *Version Number* (VER) which indicates the version of the IP in use, and hence the format of the internet header.

The *Internet Header Length* (IHL) is the length of the internet header and thus points to the beginning of the data.

The *Total Length* (TL) is the length of the datagram, including internet header and data. There are several protocol options, some of which are discussed in the next section.

4. Additional Features

The following optional mechanisms are available in the IP for use when needed.

4.1. Source Routing

The *Source Route* option provides a means for the source of a datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination.

As described above, routing at each gateway is based on the internet address in the destination field of the datagram header. If the source routing option is used, a series of additional internet addresses will be present in the option field. When the address in the destination field has been reached and the source route is not empty, the next address from the source route becomes the new destination (and is deleted from the source route list).

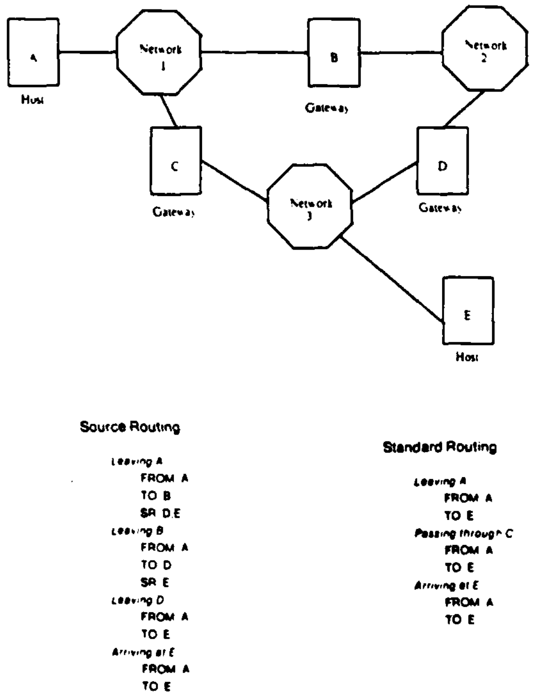


Fig. 5. Source Routing Example.

Thus, the source specifies a series of points the datagram must pass through on the way to its final destination. Normal internet routing is used to reach each of these points in turn, and the datagram may pass through a number of intermediate points between the specified addresses. Source routing may be used to specify routes to networks that are not known to the full internet system.

In Figure 5 an example of source routing is shown. Here host A is sending a datagram to host E. The normal routing would most likely be through the gateway C. We assume the user at host A would prefer in this case to have this datagram routed through gateways B and D. The Figure shows the address information at each step along the route.

4.2. Return (or Record) Route

The *Return Route* option provides a means to record the route taken by a datagram. A return route is composed of a series of internet addresses. When an IP module routes a datagram and the return route option is present, the gateway inserts its own internet address (in the environment of the next destination) into the return route option data.

4.3. Error Report

The *Error Report* option is used to report an error detected in processing a datagram to the source. A code indicates the type of error detected, and the ID is copied from the datagram in error, and additional octets of error information may be present depending on the error code. If a datagram consisting only of an error report option is found to be in error or must be discarded, no error report is sent.

Error codes are defined to report the following conditions: (0) No reason given, (1) Not Accepted – no program at the destination will accept the datagram, (2) Fragmentation Problem – the datagram cannot be delivered without fragmenting and the DF flag is set, (3) Reassembly Problem – the datagram cannot be reassembled because there are missing fragments and the time to live has expired, and (4) Gateway Congestion – the datagram was discarded to relieve congestion.

5. Gateway Functions

This section summarizes the tasks performed by a gateway; which are, interfacing to the local networks, and performing the IP functions.

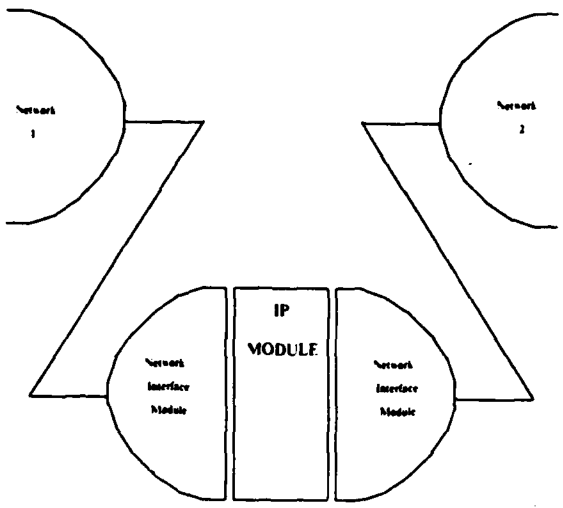


Fig. 6. Gateway.

The actual interconnection of networks is performed by gateways which are computers connected as hosts on several networks (see Figure 6). Messages are communicated across networks by using the protocols and conventions of the individual networks. While traversing each network the IP datagram is encapsulated within the local network protocols. At the gateway the IP datagram is decapsulated and examined by the gateway to determine how to route this datagram, and what local network options to use, if any. The gateway handles issues of routing, fragmentation (if the local network cannot handle regular size datagrams), error reporting and control, and interfacing to local networks.

The essential purpose of a gateway is to forward each datagram toward its destination. The key decision a gateway must make is the routing decision. When a gateway receives a datagram it must use the destination address in the IP header along with routing information stored in the gateway to determine where to send the datagram.

The routing information stored in the gateway may be relatively static (changed only by manual intervention) or dynamic (changed automatically). Both cases are allowed in the ARPA-Catenet system. The discussion of the techniques for dynamically updating the routing information are described by Strazisar [15].

Another important task of a gateway is to encapsulate datagrams for transmission through the next network, using that network's existing message trans-

]

fer protocol. This involves adding an appropriate message header (and perhaps trailer), to the datagram. The gateway must interpret the type of service field of the IP header to select the appropriate service in the next network.

The gateway decreases the TTL to account for the time elapsed since the TTL was last adjusted. This is an estimate of the time spent in transmission and processing. If this reduces the TTL to zero the gateway discards the datagram.

If the datagram is larger than the maximum packet size of the next network, the gateway may fragment it into pieces that will be sent separately.

If the gateway must discard a datagram due to congestion or errors in processing the datagram (such as an unknown or currently unreachable address), it sends an error report datagram to the source of the discarded datagram.

Of course, the gateway verifies the IP header checksum on every datagram it receives before processing it. If the check fails the datagram is discarded with no notification to the source or adjacent gateway. Since some of the IP header information is changed during gateway processing (e.g. TTL), the gateway computes a new IP header checksum before sending it on.

Each datagram can be processed completely independently of other datagrams. The provision of error recovery, sequencing, or flow control functions are left for end-to-end protocols, and the gateway does not maintain any status information or dedicate any resources for individual virtual circuits. Indeed, the gateway is unaware of any details of the higher protocol levels.

6. Design Decisions

The key decision in the design of the ARPA Internet Protocol is the choice of a datagram basis rather than a virtual circuit basis. Using datagrams as the basis of communication in the Catenet permits the use of simpler gateways since they are not required to maintain state information about the individual virtual circuits, and allows the end-to-end communication to continue via alternate routing if a gateway fails.

Using datagrams as the basic communication service allows the construction of virtual circuit style end-to-end services (e.g., TCP), and other services. In the DARPA research program there are needs for other styles of communication service. For example, the packet speech requires a service which provides minimal delay even at the cost of a few dropped messages. Such a service can be built on a datagram base, but not on a virtual circuit base. For more detail on the tradeoff between a datagram base and a virtual circuit base for communications see references [8–12].

This choice of a datagram base for the operation of the Catenet results in the separation of the internet protocol from the end-to-end protocols in general and TCP in particular. The early proposals for TCP did not focus clearly on the responsibilities of the gateways and did not allow for alternate styles of communication service. Once these needs were apparent the protocol functions were separated into distinct layers.

The decision to use the encapsulation/decapsulation technique to send the IP datagrams through local nets was made to maximize individual networks' autonomy, and to avoid the need for modifications of individual networks (particularly in the area of routing) to support internet traffic [10].

The decision to fragment datagrams in gateways as they pass from a large packet network into a small packet network, but not reassemble the fragments until they reach the destination host, allows simpler gateways and minimizes the delay in the Catenet. The alternate approach of reassembly in the next gateway is explored in reference [14].

Perhaps the most difficult design decision was the choice of the address size and structure. The size of the address field is a compromise that allows enough addresses for the anticipated growth of the Catenet yet is not an excessive overhead burden. The structuring of the address into network and host fields allows the gateways to process datagrams destined for distant networks on the basis of just the network field. This field separation also reflects an administrative delegation of the address assignment function.

In addition to the address, IP carries additional address or multiplexing information in the protocol field. This indicates which next level protocol should be used to interpret this datagram. Most of the higher level protocols have further multiplexing information called *ports* in their headers. The IP approach to addressing may be characterized as hierarchical [10].

An option in IP supports the concept of source routing. This means a source may specify a series of addresses which are used in turn until the ultimate destination is reached [10]. The decision to include this feature was motivated by the realization that many small networks may be interconnected to the Catenet via ad hoc arrangements, and destinations in such networks (or such networks themselves) may be unknown to gateways in the general Catenet.

IP uses a Time To Live which is decremented by each gateway by at least one unit (more if the datagram is delayed in the gateway for a substantial time). Other protocols use a hop count which is incremented by each gateway [5]. The practical difference is small, though the time to live approach remains effective as the size of the network changes, and allows the source to specify a maximum life for the datagram.

7. Research Issues

7.1. Multiple Addresses

There are several issues related to more flexible addressing that the current IP does not deal with. One case is a host with two (or more) internet addresses, either on one network or even on different networks. Sometimes this serves to distinguish between logically separate hosts, but in other cases it is desirable to consider both addresses as the "same place" as far as higher level protocols are concerned. It is not clear how a gateway could know when or how to route messages sent to one address to another address (e.g. if the first address was unreachable). A particularly difficult example of this problem is a mobile packet radio which moves from one network to another while trying to maintain unbroken communication.

7.2. Local Networks

A second issue is the addressing of local networks. There will soon be a large number of local networks (e.g., networks within one building or on a campus) wishing to use the ARPA-Catenet for long distance interconnection. It seems unreasonable that every one of these should have the same status as a nationwide network, with all gateways responsible for maintaining routing information about them. It may be preferable to introduce another level in the addressing hierarchy, or to combine a gateway plus internal address for such nets in the local address field of IP addresses [16].

7.3. Multiple Destinations

Another addressing issue is provision of a capability to send datagrams to a number of destinations at once. Broadcast to all is, of course, the ultimate multi-destination, but "to all" is easier to handle then "to some." This capability is inherent in the technology of some networks (e.g. satellite, ring, and Ethernets) but there is no provision in the current IP for such multidestination addressing. These is work underway in the ARPA community on an internetwork digital packet speech conferencing experiment. A protocol called ST developed for that experiment does contain a multidestination capability [17].

7.4. Naming/Addressing/Routing

The mapping of character string names that are convenient for people into internet addresses is often a problem. This can be eased by the provision of a "directory assistance" service or name server [18]. A name server is a service with a table of name/address correspondences. When the name server is sent a query about a name it responds with the name and corresponding address(es). Directory services can be provided in a centralized and/or distributed fashion. For a further discussion of the roles of names, addresses, and routes see [19].

7.5. Congestion Control

Congestion control is a problem for any network. The gateways may be viewed as nodes of the Catenet, much as IMPs are the nodes of the ARPANET. As internet traffic increases, gateways may become overloaded, even while the individual networks connecting them are enforcing their own congestion controls. Thus there may be a need for an internet congestion control mechanism which is effective with the datagram mode of operation in the Catenet. Several methods such as isarithmic control, buffer categories, and "choke" packets [20] have been proposed for such environments. The ARPA gateways implement a simple strategy of notifying the source when a packet must be discarded due to congestion.

7.6. Monitoring and Adminstrative Control

Accounting is another basic internetworking requirement. Traffic statistics are useful for monitoring and control purposes, and are easily collected by the gateways either on a net-to-net basis, or with more detail by internet source/destination pairs. Volume of packets and/or bits can be collected by a set of counters, and periodically dumped to a Catenet monitoring and accounting center. A gateway monitoring and control center is now operating to coordinate the collection of these statistics [21].

8. Conclusions

The ARPA Internet Protocol provides a common base for supporting higher level protocols in a network independent multi-network environment. The datagram basis of the internet protocol has allowed the flexible evolution of a variety of application specific higher level protocols while allowing simple gateways to interconnect networks. The principle of encapsulation for transmission through individual networks is essential for the provision of internet service over a variety of networks without requiring changes to each networks' internal operation.

As of August 1980, IP is implemented in 12 gateways interconnecting 10 networks, including packet radio, satellite, local nets, and the original ARPANET. Gateways are typically PDP 11/40 or 11/03 processors with limited memory. High level protocols including TCP, terminal access (Telnet), and file transfer (FTP) are in use above IP. Transaction oriented services such as directory assistance (Name Serv..., are also in use. Other applications are under development.

Acknowledgments

IP has developed in the context of a multicontractor research effort. It is not possible to list all the contributors that have participated in the DARPA Internet Program, however special mention should be made of the coordinating effort and technical contributions of Dr. Vinton G. Cerf of DARPA.

References

[1] Postel, J., "DOD Standard Internet Protocol," IEN 128, RFC 760, USC/Information Sciences Institute, NTIS document number ADA079730, January 1980.
[2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Defense Advanced Research Projects Agency, July 1978.
[3] Postel, J., "DOD Standard Transmission Control Protocol," IEN 129, RFC 761, USC/Information Sciences Institute, NTIS document number ADA082609, January 1980.
[4] Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Transactions on Communications, vol. COM-22, no. 5, May 1974.
[5] Boggs, D.R., et al., 'Pup: An Internetwork Architecture," IEEE Transactions on Communications, vol. COM-28, no. 4, April 1980.
[6] Shoch, J., D. Cohen, and E. Taft, "Mutual Encapsulation of Internet Protocols," Trends and Applications 1980: Computer Network Protocols, National Bureau of Standards, Gaithersbug, Maryland, May 1980.
[7] Deparis, M., et. al., "The Implementation of an End-to-End Protocol by EIN Centers: A Survey and Comparison," Proceedings International Conference on Computer Communication, Toronto, Canada, August 1976.
[8] Grossman, G.R., A. Hinchley, and C.A. Sunshine, "Issues in International Public Data Networking," Computer Networks, vol. 3, no. 4, August 1979.
[9] DiCiccio, V., et. al., "Alternatives for Interconnection of Public Packet Switching Networks," Proceedings. Sixth Data Communication Symposium, ACM/IEEE, November 1979.
[10] Sunshine, C., "Interconnection of Computer Networks," Computer Networks vol. 1, no. 3, pp. 175–195, January 1977.
[11] Cerf, V. and P. Kirstein, "Issues in Packet-Network Interconnection," Proceedings of the IEEE, vol. 66, no. 11, November 1978.
[12] Postel, J., "Internetwork Protocol Approaches," IEEE Transactions on Communications, vol. COM-28, no. 4, April 1980.
[13] Cohen, D. and J. Postel, "On Protocol Multiplexing," Sixth Data Communication Symposium, ACM/IEEE, November 1979.
[14] Shoch, J., "Packet Fragmentation in Internetwork Protocols," Computer Networks, vol. 3, no. 1, February 1979.
[15] Strazisar, V., "How to Build a Gateway," IEN 109, Bolt, Beranek, and Newman, August 1979.
[16] Cohen, D., "On Addressing and Related Issues (Or Fuel for a Discussion)," IEN 122, USC/Information Sciences Institute, October 1979.
[17] Forgie, J., "ST – A Proposed Internet Stream Protocol," IEN 119, M.I.T. Lincoln Laboratory, September 1979.
[18] Pickens, J., E. Feinler, and J. Mathis, "The NIC Name Server – A Datagram Based Information Utility," Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks, August 1979.
[19] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON Fall 78 Proceedings, (IEEE Catalog Number 78CH-1388-8C), Washington, D.C., September 1978.
[20] Grange, J., and M. Gien, eds., Flow Control in Computer Networks, North Holland, February 1979.
[21] Flood Page, D., "ARPA Catenet Monitoring and Control," IEN 105, Bolt, Beranek, and Newman, May 1979.

"INTERNETWORKING IN THE MILITARY ENVIRONMENT"

by B.H. Davies and A.S. Bates

Royal Signals and Radar Establishment
Gt. Malvern, Worcs, U.K.

Abstract

The increasing requirement for data communications in the military environment and the heterogeneous nature of the network technologies and protocols involved are highlighted. The main section of the paper discusses how the design of a military internet architecture is influenced by the military requirements especially that of survivability. Comparison with the civilian PTT approach to internetworking shows that while there are economic advantages to using civilian international standards where possible, these standards do not satisfy the military requirements. In particular the strategies for routing in a heavily damaged network environment and addressing hosts that migrate from one network to another must form an integral part of the overall architectural design. This results in gateways whose routing tables have a finer degree of detail of the internet topology than is usually required but which do not contain connection oriented information.

Finally, practical experience gained on the ARPA catenet system is described.

I. Introduction

The increasing complexity and tempo of modern warfare has rapidly created the need for flexible data communications, parallel to those associated with the "information technology" growth in the civilian environment. The aim of this paper is to highlight the differences in emphasis between data communications in the civilian and military environments, and to examine the consequence of these differences. In particular, the importance of an overall communications architecture, in order to provide survivable and interoperable communications involving both present and future systems, cannot be overstated.

Experience gained in connecting a prototype military network to the ARPA catenet system and measurements made using internetworking data transport protocols are described. Enhancements to the system to improve survivability and performance are suggested.

II. The Requirement

To a large extent, the increase in the demand for data communications stems from the increasing use of computers, microprocessors and digital circuitry in weapons, sensor, and command and control systems. These devices are used for similar reasons to those pertaining in the civilian environment, in that they can perform well specified tasks faster, more reliably and more cheaply than human personnel. However, in order to accomplish the overall goal of efficient deployment of military resources, these geographically separated devices must communicate with each other and exchange information in a hostile environment. A distinctive property of the communications between these devices, is the very "bursty" or non-continuous nature of the information transfers, which makes packet switching an attractive means of providing the communications. In packet switching, bandwidth is only allocated on demand, and therefore this technique allows considerably more efficient sharing of communication resources than the use of dedicated communication links. A further advantage of a well designed network, is the inherent survivability of communications that it provides. This does not mean that networks in a damaged condition provide the same quality of service as in their pristine condition, hence the necessity for priority markings to indicate which data is the most important. However, we can say that packet switching is an economical means of distributing the communications resources in such a manner that it is difficult for the enemy to completely destroy communications between users of the network.

So far we have described a single set of users connected to one network. However, there are many different types of networks based on different technologies and providing different types of service. This diversity of network types is due to the different user requirements and environments. For example, naval data communications may well be provided by a packet satellite network because of the large geographical area of coverage required and the great mobility of the hosts or users of the network. In the forward area tactical environment, the data communications may well be provided by a frequency hopping packet radio network, because of the extreme hostility of the electromagnetic environment. Finally, in an underground control centre, or on board a single ship, the communications may be provided by a "local area network".

Besides these different hardware technologies the grade of service provided to the user may differ. For example, a network which is primarily designed for transporting sensor information, may well be optimized for providing minimum delay in the delivery of the data, rather than providing reliability of delivery, because of the perishable nature of the data. Thus, users who are primarily interested in reliable delivery would have to initiate transport control features on an end-to-end basis, to provide for loss and misordering of the data by the network.

There is a requirement for users on the different networks to communicate with each other [1]. In particular, the long haul communications may be provided by a common bearer network, which may interconnect forward area networks with local command centre networks. Also, with additional tasks and new capabilities, there will continue to be new and unknown data communications requirements, which will have to be integrated with existing systems.

The main requirements of data communications are that they should be secure, survivable and interoperable [2]. This paper concentrates on the survivability and interoperability issues, and the reader is referred to the references which concern computer and network security [3,4]. However, it is necessary to point out that the more interoperable the systems are, the greater the security risks, because there are more avenues of attack on the confidentiality and integrity of the data, by a greater number of personnel. In particular, "access controllers" or security sentinels in critical gateways, which interconnect networks, may restrict access to certain types of traffic, thus sacrificing survivability and flexibility in the interests of security. Survivability of communications has many different meanings, but in its strictest sense it implies fully automatic routing around damaged switching components or links, and the ability to use alternate routes, even through other networks, in such a way that data integrity is maintained on an end-to-end basis.

III. Reasons for an Overall Architecture

To date, most communications systems have not been designed with an overall communications architecture in mind. This has resulted in great difficulty in providing interoperability with other systems. Because the modulation and coding, addressing and message representation, have often been combined, interconnection with another system has involved a very expensive box between the two systems. The disadvantages of this approach are:-

1) Each interface box is a special 'one off' design, which is custom built and therefore very expensive in design time and procurement cost.

2) Inevitably, in translating between one system and another, there will be certain features and services that will not have an equivalent in both systems.

3) Because of the processing power required to translate at all protocol levels, the interface unit will be a large and expensive piece of hardware. This has an effect on survivability, in that because the interface units are expensive, the minimum will be procured and the survivability of the overall communications will be determined by these vulnerable interface units.

The problem of deciding on the best architecture for computer to computer communications, has been the subject of sustained discussion over the past decade. In particular, the International Standards Organization's subcommittee 16 has produced a major document in this field, "Reference Model of Open Systems Interconnection" [5]. The central thesis of this document is that the most flexible architecture is a layered one, in which each layer has a well specified function and provides a well specified service to the layer above it. In particular, any given layer views the layers below it as a single entity. This is analagous to structured programming, where the user of a procedure call is only interested in how parameters are passed to and from the procedure and not in the internal structure of the procedure. The seven layer model is illustrated in figure 1. Two points about the model are relevant to the discussion below. Firstly, the functional specification of each layer is more difficult to agree on, the higher the layer, because in these layers in the architecture there are more choices. Secondly, there has as yet been no ISO agreed protocols for implementing any of the layers. The model itself does not preclude more than one protocol implementing a given layer of the architecture.

IV. Current State of Civilian Standards

In Europe, with its highly regulated public communications authorities, there has been a very active co-operation among various countries to establish data communications standards from the outset. The CCITT (The International Telegraph and Telephone Consulative Committee), which is the corporate body representing the telecommunications authorities of these countries, has developed standard protocols, X25 [6], for levels 1,2 & 3 of the ISO reference model. It is important to note that in arriving at these standards, the PTTs (Public Telegraph and Telephone authorities) have identified that most customers want a connection orientated type of service, ensuring ordered and reliable delivery of packets. The network reserves the right, in event of a network error or congestion, to send a reset to both ends, indicating loss of data integrity. At present, no figures are available to indicate the frequency of such events. Because the main public networks in Europe are X25 networks, there has been considerable pressure on computer manufacturers to provide X25 hardware and software products off the shelf. This has led manufacturers of private networks, in particular local area networks, to consider providing X25 accesses, in order to facilitate connections to existing machines and operating systems. Thus, X25 is rapidly becoming a de facto international standard in Europe.

What about the interconnection of X25 networks? Obviously, connecting networks which use the same access protocols and provide the same grade of service, is not so difficult a problem as interconnecting very dissimilar networks. Thus, there are X series protocols, X75, X121 [6], which enable PTT's to provide connections between users on different X25 networks, and although not all X25 facilities are available on internetwork connections, the service offered is analagous to STD dialling of international telephone calls. However, these protocols do rely on the X25 networks themselves, to route the internet packets to the gateways. It appears that private networks will not be allowed to connect to public networks via X75 gateways, and so gateways between private and public networks will have to provide a service between two X25 calls back-to-back, and will thus act as a staging post for the user's data.

Protocols for the transport layer (layer 4 of the OSI Reference model), are not so well developed as for the lower layers. However, in the United Kingdom a transport protocol [7] has been defined, and implementations above X25 have been realized. The most notable feature of this protocol is the flexible addressing structure, which allows connections to be established across different naming/addressing domains.

Before considering the applicability of these developments in the military environment, it is useful to consider some of the differences in emphasis, between civilian and military networks, and their usage.

V. Comparison Between Civilian and Military Networks and their Usage

1) The usage of military networks in time of war is very difficult to predict. Although major exercises give some idea of the user demand, past experience has shown that these are slightly artificial and may not give a true picture. In civilian networks, usage can generally be accurately predicted by extrapolating present useage patterns, with economic and equipment sales factors being taken into account.

2) The availability of the full capacity of a military network may well be degraded when it is most needed, because links may be jammed and nodes and gateways physically destroyed. In the civilian environment, there is usually a very high availability of hardware and data links, with the use of standby power supplies and 'hot' spares for critical nodes such as gateways.

3) In general, there is a considerably higher degree of mobility of both users and networks in the military environment. In particular, airborne networks such as JTIDS (Joint Tactical Information Distribution System), with users such as fighter aircraft, will place stringent requirements on internetwork connections and survivability. A consequence of this will be that the users may well be completely unaware of the internet topology. While mobile access to networks will obviously develop in the civilian environment, in general it constitutes a fairly static community of networks and users.

4) One of the major advantages of geographically distributed databases, which are flexibly interconnected with communications links, is the decrease in vulnerability of the overall system to the total failure of a site (eg by physical destruction). Thus, when designing military networks it is important not to introduce an Achilles heel by, for example, employing a centralized network control centre. However, centralized control may well be the most convenient and cost-effective solution in civilian environment.

5) Both civilian and military network authorities wish to provide secure, survivable, interoperable, and guaranteed grades of service to their users. The questions arise as to how much the user is willing to pay for these properties, and how important the properties are? The question of the importance of the property, depends on the threats to the network, and these are obviously substantially greater in the military case. This means that the solutions for military networks may well be more expensive, in terms of implementation and running costs, than those for the civilian environment.

VI. Techniques For Network Interconnection

At present there are two main architectural methods [8] for providing process to process communication across dissimilar networks. They are referred to as the "end-to-end" and "hop-by-hop" methods, because in the former, all the control information relevant to a particular data connection is held only in the source and destination hosts, while in the latter, connection oriented information is also held in various intermediate switching nodes, called gateways.

The end-to-end approach is based on the assumption that all networks will offer at least an unreliable datagram service, ie if a sequence of packets is injected into the network then the destination will receive some of them, possibly misordered, and with possible duplication. Any improvement on this grade of service will be achieved by implementing end-to-end procedures to perform reordering, retransmission of losses and detection of duplicates. A legitimate criticism of this approach is that these upgrading procedures are acting across all the networks in the chain, which in the case of good networks means that there are extra overheads which involve needless expenditure. Thus, in the hop-by-hop approach, the required level of internet service is provided by procedures implemented across each network. This is obviously more expensive initially, in that the procedures are different for the different networks, but its running costs are cheaper because unnecessary control and retransmissions do not occur across the networks providing the higher grade of service.

There are also two schools of thought on addressing strategy, which are difficult to completely separate out from the ideas set out above. The first school, which has to date been associated with the end-to-end approach, is that all networks worldwide should have a unique network number allocated to it by a global authority. Thus, any host address can be uniquely defined worldwide by concatenating its network number with its host number. The addressing of internet packets is then simple. The other school believes that such international agreement on address formats is not achievable in the near future, and that there will exist multiple naming/addressing authorities. Thus, the address field will have to consist of a list of addresses in different formats, which will be parsed by the gateways of the different naming authorities as the packet wends its way through the internet system. This second system is considerably more flexible than the first, but as we shall see has other consequences as well.

To date, operational systems of the end-to-end variety have used a flat addressing space and the hop-by-hop systems have used the multiple domain system. A schematic representation of the protocol layering involved, in an internetwork connection across three networks, is shown for both the hop-by-hop and end-to-end approaches in figure 2. The hop-by-hop diagram clearly illustrates that the total service is provided by three concatenated services, involving different transport protocols on different types of networks. The end-to-end representation illustrates the singular nature of the transport service, which is independent of attributes of the underlying networks. We will now compare the advantages and disadvantages of the two systems, in the light of operation in the military environment.

1) Running Costs The hop-by-hop approach has the advantage over the end-to-end approach as far as the civilian user is concerned, in that it is very 'tariff' conscious (ie it only uses the minimum amount of transport protocol necessary to provide the required grade of service). Now as many of the European networks provide the high reliability of a virtual call service, this means that hop-by-hop implementations of the transport service for these networks will involve minimum overheads in te ms of extra bits to be transmitted, and therefore their running costs will be minimal.

In the end-to-end approach, every packet carries a full internet source and destination address in its header, so that it can make its own way to its destination. In the hop-by-hop approach once the call has been set up, only the destination address for that particular network has to be carried, because the gateways on route contain addressing information for further hops.

2) Development Costs The philosophy of the hop-by-hop approach implies a different protocol for each different type of network. This is not so serious in the civilian environment, because of the considerable influence of the CCITT standards which means that most European public and private networks are of the X25 variety. Even local networks with very high speed interfaces are planning to implement an X25 access. However, in the military environment, where there is a considerably greater range of networks, this could require the development of a number of transport protocols.

3) Trusting Transit Networks When a user makes a multi-net connection, using the hop-by-hop approach, it implies that he trusts the level of transport service being offered by the intermediate gateways in the internet route. Furthermore, it implies that he is happy with the reliability of intermediate gateways whict, albeit temporarily, take responsibility for his data at the termination of each hop. We believe that this is a state of affairs that is considerably more acceptable in the benign civilian environment than in the hostile military one.

In the end-to-end approach, only an unreliable datagram delivery service is expected from the set of concatenated networks, and loss of data in any intermediate switching node or gateway will be recovered by a retransmission from the source. Therefore, maintaining the bit integrity of the data transmission does not rely on the continuing correct operation of an intermediate node.

4) Addressing Strategy In the multi-domain address strategy, if a user in one domain wishes to communicate with users in another domain, the user must know the topology of the interconnection of these domains, so that he can supply the information necessary for his data to reach the destination domain. This information could be obtained automatically for him, but it implies separate and possibly different bilateral agreements between the various domain authorities.

In the end-to-end approach with a flat addressing space, each packet contains complete addressing information, and is free to find the best current route across all intermediate networks (figure 3). This dynamic internet routing has similar resource allocation advantages to dynamic routing on single networks. This flexibility of routing in the internet environment is more important in the context of the more rapidly changing scenario of the military environment.

5) Transport Control The end-to-end control is certainly less flexible than the hop-by-hop control. Timeouts in particular, may vary by an order of magnitude, even on the networks in service today. End-to-end flow control, also requires more sophisticated strategies than are needed in the hop-by-hop method.

6) Gateway Complexity One of the chief attractions of the end-to-end approach with flat addressing is the conceptual simplicity and relative smallness of the gateways with respect to the hop-by-hop approach. This is because the only modules that vary from gateway to gateway are the network access modules that pertain to each network (and these are just the modules needed on all hosts attached to that network). The fact that no connection oriented information is held in the gateway, greatly simplifies the action that the gateway has to take on receiving a packet and the amount of buffer storage it needs. This property ties in well with the gateway policy for military networks, namely that networks should be multiply connected by

gateways in order to provide survivable internetwork communications. Thus, the "simplicity" of the gateways will result in cheapness and the ability to provide more than one gateway between every pair of networks.

Thus, although the end-to-end approach involves higher overheads in terms of packet headers we believe that it offers considerably increased survivability in a hostile environment. Furthermore, in a situation in which users and networks are mobile, it is necessary for all networks to come under a single naming/addressing authority (eg NATO) if these changes in topology are to be distributed rapidly and efficiently throughout the internet system.

VII. The ARPA Catenet System

An example of the end-to-end approach with a flat address space, which has been running operationally for about 5 years, is the ARPA catenet system. This system connects about thirty different networks including land-line, satellite and radio based networks, as well as a variety of local area networks. The thinking and concepts involved in the architecture of this system have been fully described in a number of papers [9, 10].

The protocols responsible for data transport in this system and their hierarchical relationship are shown in figure 4.

1) Internet Protocol (IP) [11] This provides for transmitting blocks of data, called datagrams, from sources to destinations. Its main parameters are source and destination addresses which are globally unique. Implementations of this protocol exist in the gateways and internet hosts. The datagrams are routed from one internet module to another through individual networks. In this approach, datagrams may be routed across networks whose maximum packet size is smaller than they are. In this case, a fragmentation module breaks up the packet into smaller packets, replicating enough information in the headers to allow reassembly at the destination. Reassembly does not take place in the gateways, because packets may take different routes to their destinations. There are a number of options available in the internet protocol and these are specified in the control information of the header. Thus, the internet header is of variable length.

2) Transmission Control Protocol (TCP) [12]. TCP is a data transport protocol appropriate to level 4 of the ISO reference model, and is especially designed for use on interconnected systems of networks. TCP is a connection oriented, end-to-end reliable protocol, designed to fit into a layered hierarchy of protocols which support multi-network applications. It provides for reliable interprocess communications, between pairs of processes in host computers, attached to distinct but interconnected computer communication networks. The TCP assumes it can obtain a simple, potentially unreliable, datagram service from the lower level protocols. It fits into a layered protocol architecture just above a basic Internet Protocol, which provides a way for the TCP to send and receive variable length segments of information enclosed in internet datagram envelopes. In order for the TCP to provide a reliable logical circuit between pairs of processes, on top of the less reliable internet communication system, it performs the functions of basic data transfer, data acknowledgement, flow control and multiplexing.

3) Gateway to Gateway Protocol (GGP) [13]. The gateway to gateway protocol is responsible both for distributing routing information through the gateways of the catenet and for advising communicating hosts of routing changes, congestion control and unreachable destinations. The basic routing algorithm, in use today, is the original ARPAnet routing algorithm. This involves gateways telling their nearest neighbours which networks they can reach and how many gateway to gateway hops are involved in the route. If a gateway is directly connected to a network, then it is said to be zero hops to that net. Gateways continuously monitor the state of the network access switch to which they are connected and their nearest neighbour gateways to ensure that routes through them are still available.

VIII. Practical Experience of the ARPA Catenet System

In the autumn of 1978, RSRE set up a collaborative program of research and development in communications with the Advanced Research Projects Agency of the US Department of Defense. This collaborative program involved the connection of the PPSN (Pilot Packet Switched Network), our own in-house research network, to the ARPA catenet system, and providing terminal and file access from an internet host on the PPSN to some of the major ARPAnet hosts. The first two years of the program were allocated to the development and implementation of a reliable connection between PPSN and the ARPA catenet system. We have implemented the DoD standard Transmission Control Protocol, the Internet Protocol and the Gateway-to-Gateway Protocol in Coral 66. In addition, we have made many measurements on the performance of the catenet system, particularly in terms of round-trip delays as the connectivity and the development of the catenet has evolved.

The current configuration is shown in figure 5. The RSRE internet host (PDP-11/23) contains the standard internet protocols of Telnet, TCP and IP, and which run under our own virtual memory operating system EMMOS [19]. The link level protocol, X25 level 2, is used to interface to the PPSN. This protocol is implemented on a microprocessor communication interface (X25 line unit) which is connected to PDP-11 hosts via a standard interface [18].

The PPSN is connected to the rest of the catenet via the RSRE gateway. The gateway (PDP-11/23) has three network interfaces on it, each using a X25 line unit. They are used to provide, 1) access to PPSN, 2) a test port which can be directly connected to a measurement host, and 3) an interface which connects the RSRE gateway to a gateway at University College, London (UCL) via a 9.6k bits/s Post Office line.

The UCL gateway is connected to two other networks, 1) UCL net and 2) Satnet (ARPA packet satellite network). The connection to Satnet is via the Goonhilly SIMP (Satellite Interface Message Processor). Packets destined for Arpanet are forwarded by the Goonhilly SIMP, over the shared 64k bits/s half duplex satellite channel to the Etam SIMP, and from there they are forwarded on to the BBN gateway, and hence into Arpanet.

Catenet Measurements. Some of these measurements were made by echoing packets off the various catenet gateways (figure 5), and a small but representative sample are listed below. By time stamping the packets as they leave the measurement host at RSRE, and then comparing the time stamps with the local time when the packets return, having been echoed off the gateways, the single round trip delay is measured. These delays not only include network transition times, but also any internal delays in the gateways.

The round trip delays from the RSRE measurement host to various gateways, for internet packets containing 6 data bytes are:-

Gateway	Mean Delay (secs)	Min/Max Delay (secs)
RSRE	0.2	0.2
UCL	0.35	0.35 - 0.4
BBN	2.0	1.5 - 3.8
SRI-PRI	2.5	1.9 - 3.8

The results for the RSRE and UCL gateways correspond to the theoretical delays expected due to line speeds. The results for the BBN and SRI-PRI gateways are due to the longer satellite delays and control algorithm of Satnet.

Retransmissions in TCP. TCP can be used for communications over a variety of different networks, therefore the wide variation of round trip delays, as shown above, means that a fixed retransmission period is not suitable, since in some cases there will be significant delays when a TCP segment is lost, while in others there will be unnecessary retransmissions.

To overcome this problem we have implemented a dynamic timeout algorithm for use in TCP. This algorithm measures the time elapsed between sending a data octet with a particular sequence number, and receiving an acknowledgement that covers that sequence number. Using that measured elapsed time as the round trip time (RTT), we compute a smoothed round trip time (SRTT) as:

$$SRTT = (ALPHA * SRTT) + ((1 - ALPHA) * RTT)$$

and based on this, compute the retransmission timeout (RTO) as:

$$RTO = \min \{BOUND, BETA * SRTT\}$$

where BOUND is an upper bound on the timeout (eg 0.9), and BETA is a delay variance factor (eg 1.5).

The performance of this algorithm has shown to be very good and has significantly reduced the number of unnecessary retransmissions.

IX. Enhancements To The Catenet System

There are a number of situations, peculiar to the military context, which are not catered for by the algorithms presently used in the catenet. Before discussing these and possible enhancements to the catenet which would improve its survivability in the military environment, we must introduce the concepts of "partitioned networks" and "source routing".

A "partitioned network" is one that is so badly damaged that there exists no paths between certain of its switching nodes. Typically, this results in two or more subsets or partitions of nodes, within which communications are possible, but which cannot communicate with each other. Hosts connected to different partitions cannot communicate in the usual way. However, if this network is connected by more than one gateway to the catenet system and there is at least one gateway on each partition, hosts could still communicate by an internetwork path as illustrated in figure 6. The concepts of routing to partitioned networks are concerned with automatic and efficient routing of packets under the conditions mentioned above.

The principle of "source routing" is one of providing some of the routing intelligence in the packet header, by providing not just the destination address, but also some or all of the intermediate node addresses through which the packet has to pass. This facility is provided as an option in the present DoD Internet Protocol.

1) Changes to the Catenet Routing Algorithm. The catenet system as presently configured, permits routing around damaged networks and gateways. It assumes that hosts know the addresses of their local gateways, and are prepared to poll these gateways to determine their status, and have procedures for using alternate gateways, if the primary one is congested or inoperative. Presently, routing to a partitioned network would involve knowing the topology of the catenet and inserting the routing information in the packet header in the form of a source route. This is perfectly feasibly, but in a fast changing military environment it would be preferable if the gateways contained enough information to perform automatic routing to hosts on partitioned networks.

If the internet system of gateways is regarded as a super-datagram network, whose node to node protocol is the Internet Protocol, then it would seem reasonable that the internode routing be based on gateway or node identifiers. The routing information distributed to gateways should permit routing to a specific gateway, rather than to a network. As there may be more gateways than networks, this will involve the storage of more information in the gateways than at present. However, if there are additional gateway nodes for providing survivability it is a waste of resources if the information is not disseminated and used when most needed.

There are two reasons for wishing to change the present catenet routing algorithm:-

(i) The present algorithm suffers from oscillations when certain link failures occur, because it uses repeated minimization to compute the shortest path. Presently, this problem is overcome by having a narrow range of link costs.

(ii) The granularity, or fineness, of the information distributed by the present algorithm which performs routing to networks, is insufficient for automatic routing to partitioned networks. This is because the route into a destination net via two different gateways may be wildly separated, as illustrated in figure 7. If the network is partitioned, we need to specify the entry into the net rather than just the net.

A recognized candidate for the improved routing algorithm is a modification of the New Arpanet Routing Algorithm [14], which is currently used on Arpanet. Using this algorithm, all the gateways broadcast information to all other gateways using a flooding technique. In particular, two types of information are disseminated:-

(i) Each gateway broadcasts the names of the nets to which it is directly connected.

(ii) Each gateway broadcasts the names of its neighbours with which it can communicate.

From this information, all gateways can determine which networks are partitioned, because a partitioned net will have two or more gateways attached to it which are unable to communicate. Having implemented this algorithm, there are one or two additional techniques that are necessary for dealing with routing to partitioned networks. The main remaining problems are, determining the partition in which the destination host is located, and specifying this in packets to be sent to that host. Now specifying the partition could be accomplished by specifying the identifier of the gateway through which the partition communicates with the rest of the catenet. However, at present there is no format for specifying gateway identifiers in the internet header. The determination of which partition the destination host is in, is best done by the gateway connected to the source host's network. This gateway will know how many partitions the destination network is divided into, and the entry gateways to these partitions. When the connection is being set up, the opening packets will be sent to all partitions, and the resultant reply will contain the relevant partition identifier. A minor expansion of the internet header will be required for specifying gateway identifiers in the internet packet headers.

2) Mobile Hosts in the Military Environment. There are already a number of requirements for aircraft flying from one tactical net to another, to be able to maintain communication with a ground based command and control centre [15]. There has been considerable discussion on possible solutions to this problem [16,17]. The solution should, if possible avoid using a centralized database, not only because of its vulnerability but also because a separate communication must be successfully performed with the database, as a pre-requisite for a successful connection to the mobile host. Furthermore, as the host moves from one net to another, updates to the database must be made in a timely manner. Obviously, a third party has to be involved if two mobile hosts wish to communicate. However, the ground control centre is a natural anchor for mobile communications, and if the TCP connection identifiers were divorced from physical addresses, the scheme below would provide total data integrity as the mobile host changed networks.

An interesting point, that is immediately highlighted when considering this problem, is that the unique identification of a TCP connection is at present tied down to physical addresses. We believe that this is undesirable, and has led to the present restricted attempts at solving this problem. We believe that unique TCP identifiers should be exchanged at the start of the connection and that these be used throughout, so that any changes in the physical addresses can be exchanged without closing the connection (ie when the aircraft changes nets it inserts its new address in the source address field, this is then used by the ground to continue the connection). It is possible that there will be a little hiccup as the change over from one net to another occurs, because packets may arrive out of order, however retransmission would take care of this. It would obviously be the responsibility of the mobile host to 'login' to the ground centre on entering a network, so that a connection could be opened up from the ground. An alternative approach would be to include another protocol layer directly above the TCP layer. This new protocol would be responsible for opening and closing TCP connections and maintaining data integrity as the mobile host moved onto another network. The disadvantage of this approach, is the necessity to transfer the mobile host's new address on a three way handshake basis, before the host moved onto the new network.

3) Congestion Control in the Catenet. The catenet is essentially a super datagram network, and congestion control consists of using all possible routes to the best advantage and being able to offer a graceful degradation of service when the users demand exceed the network resources. It is important that fairness is exercised in providing a service to users, assuming that they are of the same priority. The above implies that the cost of a route should change if substantial queues build up on it, so that alternate routes become preferable in an SPF (Shortest Path First) routing algorithm. The change in cost will be reflected in the routing updates, and alternate less congested routes will be preferred. This requires a more realistic measure of internet routing costs, than the number of gateway hops used at present. This needs to be implemented on the catenet for realistic trials,

even though the numbers of alternate routes is very small. Having thus made the best use of the internet resources, the only remaining action is to throttle off users when, by their weight of numbers, they overload the system. This throttling must be fair, bearing in mind priorities. One aspect of the fairness problem is that gateways handle packets on an independent datagram basis and are not therefore conscious of "greedy" users disobeying advisory flow control messages. A full solution of this problem would require a complex control theory model to be solved. This would involve the knowledge of the queuing sizes and delays on all intergateway links. The despatching of packets from the initial gateway would only occur when its journey through the system could be undertaken without it exceeding a specified delay band.

X. Summary

Many of the concepts presented in this paper have been widely discussed in the ARPA internet community. The authors wish to thank their colleagues in the ARPA internet community for many discussions on the concepts presented in this paper.

XII. REFERENCES

[1]. G.A. LaVean, "Interoperability in Defense Communications", IEEE Trans.Comm, vol com-28, no 9, pp 1445-1455, Sept 1980.

[2]. F.F. Kuo, "Defense Packet Switching Networks in the US", Interlinking of Computer Networks, pp 307-313, NATO ASI, Bonas Sept 1978.

[3]. R.B. Stillman and C.R. Defiore, "Computer Security and Networking Protocols", IEEE Trans.Comm, vol com-28, no 9, pp 1472-1477, Sept 1980.

[4]. D.H. Barnes, "Provision of End-to-end Security for User Data on an Experimental Packet Switch Network", IEE 4th Intnl. Conf. on Software Engineering for Telecommunications Switching Systems, Warwick July 1981.

[5]. Reference Model of Open Systems Interconnection, ISO/TC97/SC16/N227, International Standards Organization, 1979.

[6]. CCITT Recommendations X Series "Public Data Networks", Orange Book, ITU, Nov 1980.

[7]. British Post Office User Forum, "A Network Independent Transport Service", Feb 1980.

[8]. J.B. Postel, "Internet Protocol Approaches", IEEE Trans.Comm, vol com-28, no 4, pp 604-611, April 1980.

[9]. V. Cerf and R. Kahn, "A Protocol for Packet Network Interconnection", Comput. Networks, vol 3, pp 259-266, Sept 1974.

[10]. V. Cerf "DARPA Activities in Packet Network Interconnection", Interlinking of Computer Networks, pp 287-313, NATO ISO, Bonas, Sept 1978.

[11]. DARPA, "DOD Standard Internet Protocol", IEN-128, Defense Advanced Research Projects Agency, Jan 1980.

[12]. DARPA, "DOD Standard Transmission Control Protocol", IEN-129, Defense Advanced Research Projects Agency, Jan 1980.

[13]. V. Strazisar, "How to Build a Gateway", Internet Experiment Note 109 Aug 1979.

[14]. J.M. McQuillan, I. Richer and E.C. Rosen, "The New Routing Algorithm for the ARPAnet IEEE Trans.Comm, vol comm-28, no 5, pp 711-719, May 1980.

[15]. V.G. Cerf, "Internet Addressing and Naming in the Tactical Environment", Internet Experiment Note 110, Aug 1979.

[16]. C.A. Sunshine and J.B. Postel, "Addressing Mobile Hosts in the ARPA Internet Environment" Internet Experiment Note 135, March 1980.

[17]. R. Perlman, "Flying Packet Radios and Network Partitions", Internet Experiment Note 146, June 1980.

[18]. A.F. Martin and J.K. Parks, "Intelligent X25 Level 2 Line Units for Switching", Data Networks: Development and Uses, Online Publications Ltd., pp 371-384, 1980.

[19]. S.R. Wiseman and B.H. Davies, "Memory Management Extensions to the SRI Micro Operating Systems for PDP-11/23/34/35/40", Internet Experiment Note 136, May 1980.

[20]. B.H. Davies and A.S. Bates, "Internetworking in Packet Switched Communications: First Report on the RSRE-ARPA Collaborative Program" RSRE Memorandum No 3281, July 1980.

Application Layer	User Programs or Processes that wish to exchange information
Presentation Layer	Concerned with data formats of information exchanged
Session Layer	Concerned with synchronization and delimiting of information exchanges
Transport Layer	Provides a universal data transport layer independent of the underlying network
Network Layer	Provides network access and routing
Link Layer	Provides data transmission over a potentially unreliable link
Physical Layer	Specifies electrical signalling for data and control of the physical medium

FIGURE 1 ISO MODEL FOR OPEN SYSTEM INTERCONNECTION

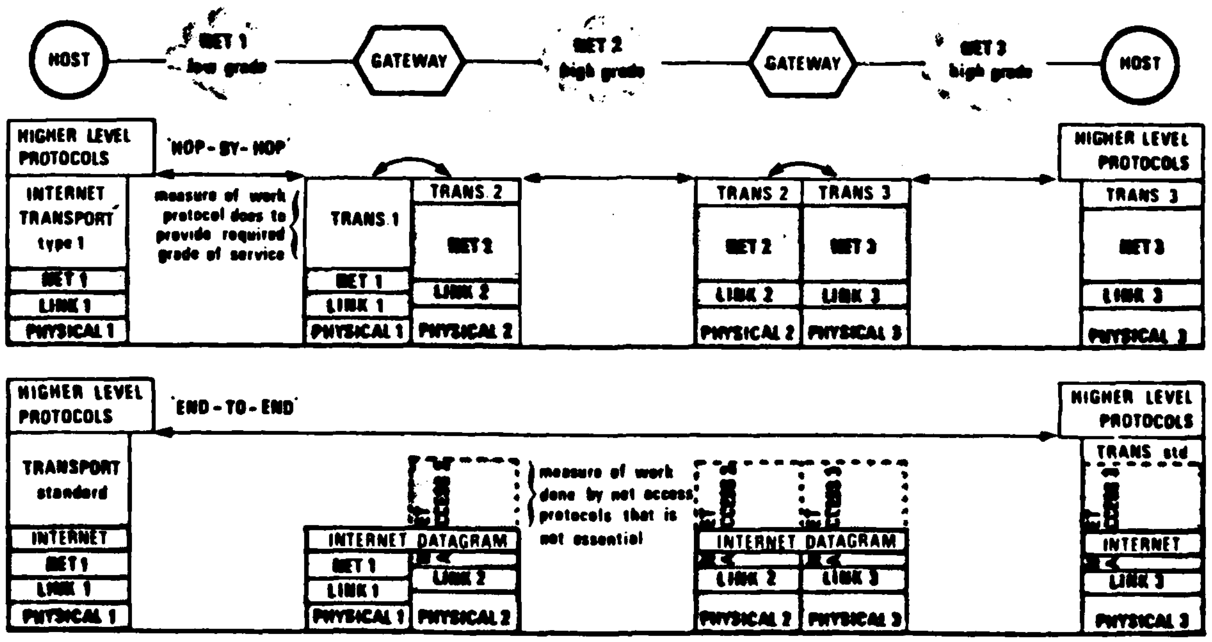
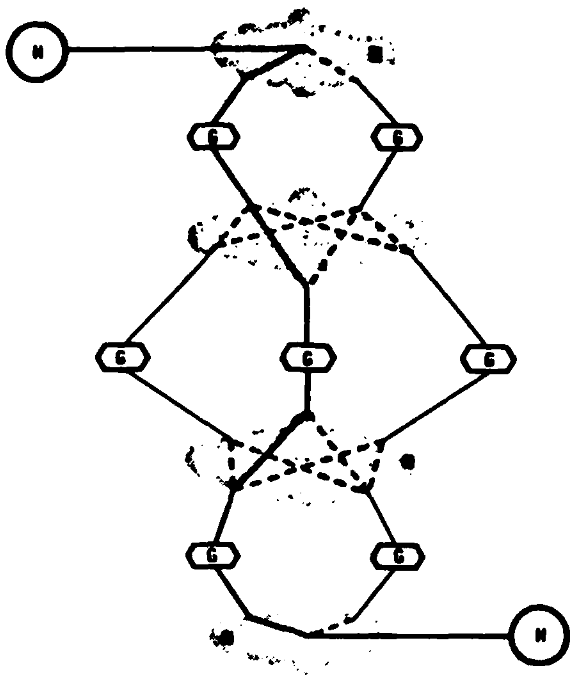


FIGURE 2 REPRESENTATION OF HOP-BY-HOP & END-TO-END PHILOSOPHIES OF INTERNETWORKING



Internet datagrams in the "end-to-end" approach may take any of the dashed or the solid routes, but data in the "hop-by-hop" approach takes the solid route set up when the connection was established

FIGURE 3 ROUTING FLEXIBILITY OF END-TO-END APPROACH

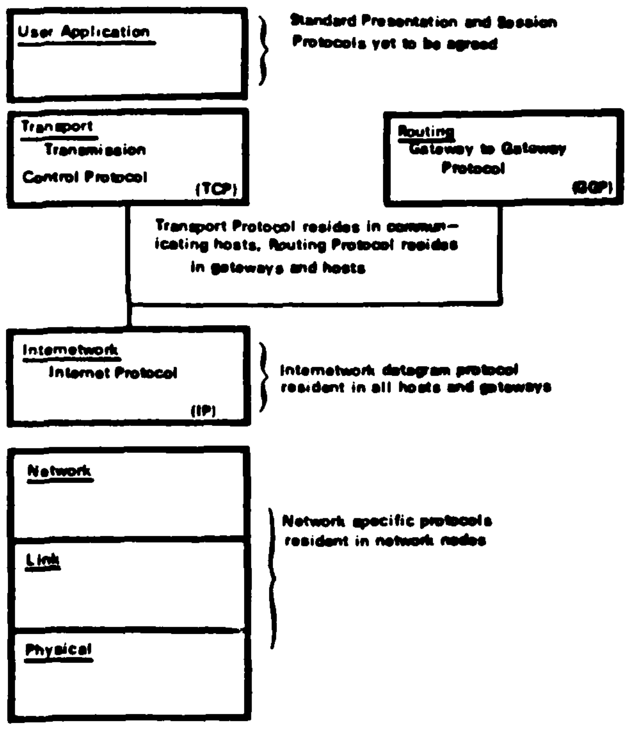

User Application
Standard Presentation and Session Protocols yet to be agreed
Transport
Transmission Control Protocol (TCP)
Routing
Gateway to Gateway Protocol (GGP)
Transport Protocol resides in communicating hosts, Routing Protocol resides in gateways and hosts
Internetwork
Internet Protocol (IP)
Internetwork datagram protocol resident in all hosts and gateways
Network
Link
Physical
Network specific protocols resident in network nodes

FIGURE 4 PROTOCOL LAYERS IN INTERNETWORKING

(116)

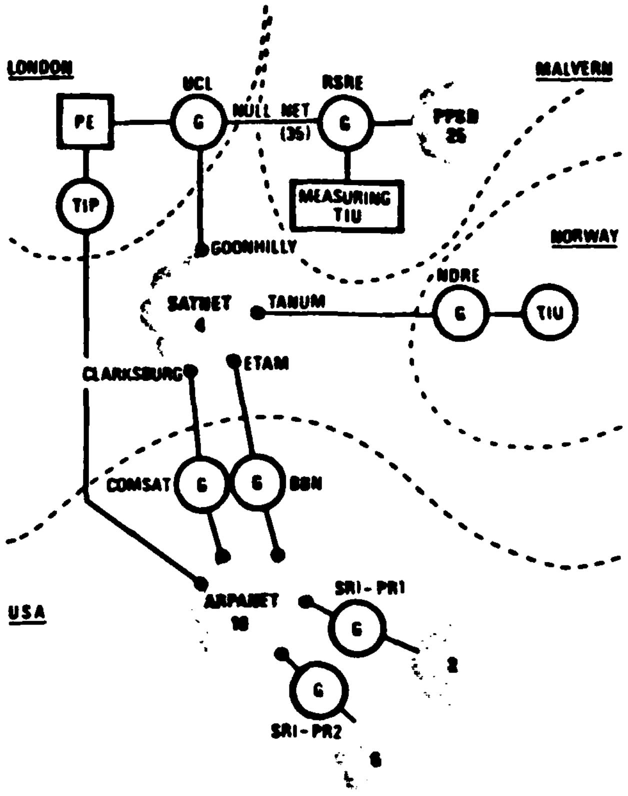


FIGURE 5 MEASUREMENTS CONFIGURATION
JANUARY 1982

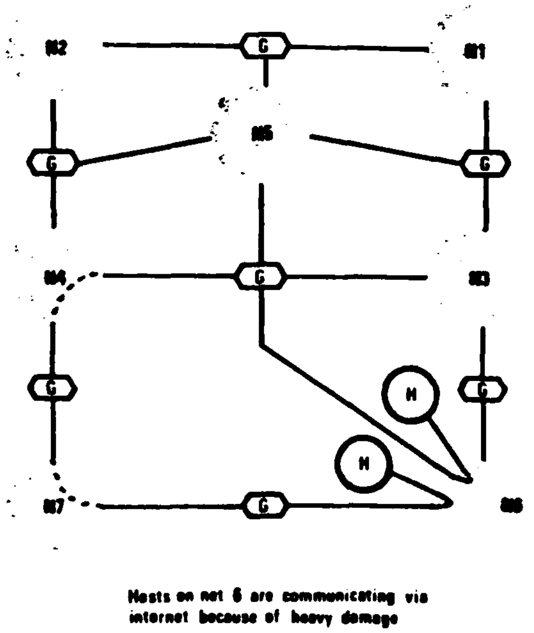

N2
G
N1
N5
N4
N3
H
N7
N6
Hosts on net 6 are communicating via internet because of heavy damage

FIGURE 6 INTERNET SYSTEM CAN PROVIDE
INCREASED COMMUNICATIONS
SURVIVABILITY

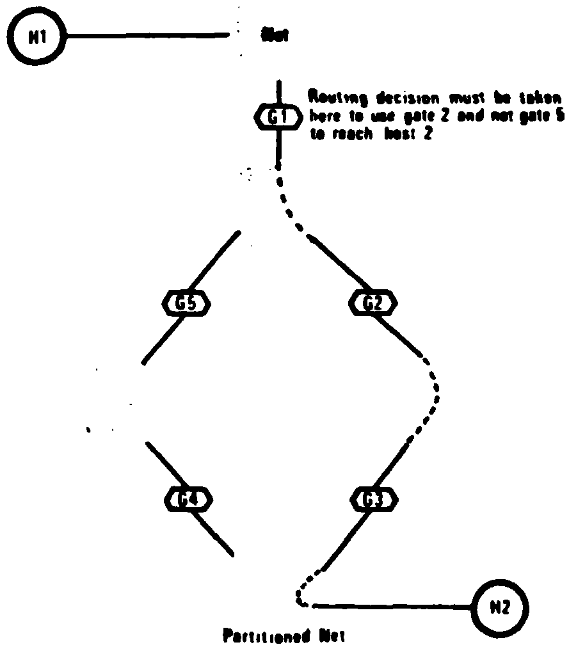
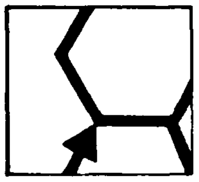

H1
Net
G1
Routing decision must be taken here to use gate 2 and not gate 5 to reach host 2
G5
G2
G4
G3
H2
Partitioned Net

FIGURE 7 ROUTING TO PARTITIONED NETWORKS

Alan Sheltzer, Robert Hinden, and Mike Brescia,
Bolt Beranek and Newman Inc., Cambridge, Mass.

Connecting different types of networks with gateways

Darpa's Internet connects more than 20 networks by gateways, which transmit datagrams and allow adaptive routing.



Just as packet-switching technology matured and spread to commercial applications, internetworking technology is now moving from the research environment into the commercial world. Gateways are being built to interconnect X.25 public packet-switching networks, and many more are planned to link various local networks such as Ethernet.

One of the original interconnected group of networks is the Department of Defense Advanced Research Project Agency's (Darpa) Internet System. It uses communications processors as gateways to link more than 20 networks that use diverse technologies.

The Internet System has been a focal point for internetworking development, with much of the technology supplied by Bolt Beranek and Newman (BBN) of Cambridge, Mass. For example, the Internet gateway transmits information in the form of datagrams and allows different routing schemes to be determined dynamically depending on the best available path. The alternative approach to the datagram model for gateways is the virtual-circuit approach, which determines and establishes a route before information is transmitted. Each scheme has advantages and disadvantages related to congestion, reliability, and overhead.

In general, gateways extend network users' abilities to access remote machines, transfer files between different vendors' computers, and send electronic mail. They also provide a solution to the problem of deciding which of the many networking methods is best by allowing all of them to be used, depending on the application. The different types of networks can then be interconnected by gateways, thus giving the user a view of only one large network configuration.

The fundamental technology of gateways is straightforward. For example, two networks "A" and "B," composed of hosts, nodes, and lines, are linked by connecting a communications processor that runs internetworking software to each of the networks. The combination processor and software is a gateway. Host A can send a message to host B on network B by first sending the message to the gateway. The gateway then forwards the message through network B to destination host B.

Several gateways can be used to interconnect a number of different networks. These multiple gateways provide redundancy and additional load capacity.

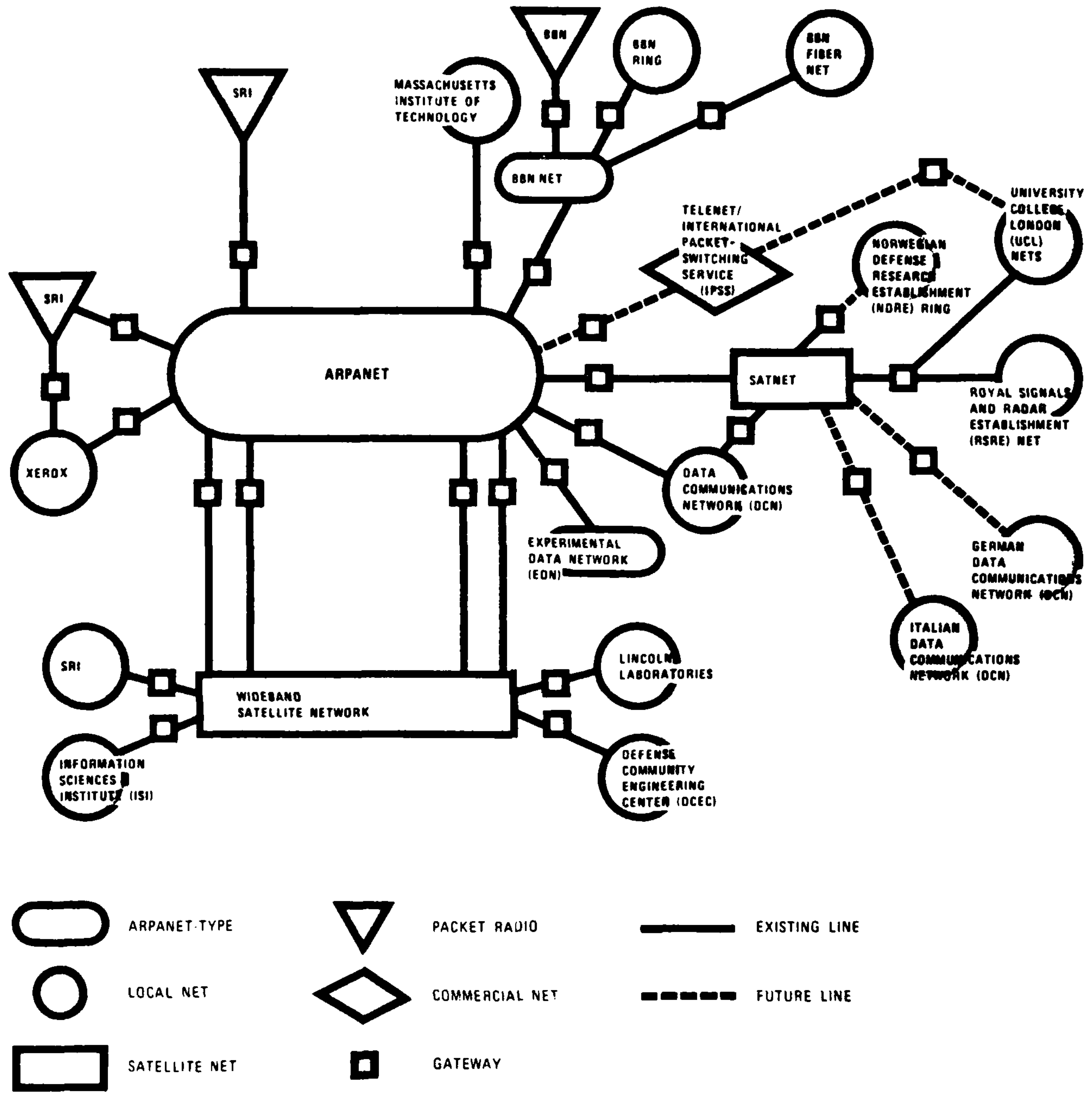
The user view of the interconnected networks is simplified if the gateways are regarded as switching nodes and the networks as lines. Then the entire configuration can be viewed as a single network, built from a collection of separate networks.

Gateways forward messages across networks to other gateways within an internetwork system just as switching nodes forward messages across lines to other switching nodes within a single computer network. However, to provide an efficient, reliable communications service, the gateways should also provide switching node functions such as adaptive routing, flow control, and network monitoring.

The transatlantic connection

There are two approaches to internetworking: the virtual-circuit approach and the datagram. In the architecture that the International Consultative Committee for Telegraphy and Telephony (CCITT) recommends, the internetwork switching nodes provide virtual-circuit service between networks. To do this, each switching node, called an X.75 gateway, is directly connected to X.75 gateways on other networks. When a call is established between two networks, virtual circuits are set up between the source host and an X.75 gateway on the source network, between neighboring X.75

1. The Darpa Internet core. *There are more than 20 networks and gateways, several hundred host computers, and several thousand terminals that make up the Darpa Internet System. The networks are connected in a distributed fashion with multiple paths between networks and alternate paths that span other networks.*


BBN
BBN RING
BBN FIBER NET
SRI
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
BBN NET
TELENET/ INTERNATIONAL PACKET-SWITCHING SERVICE (IPSS)
UNIVERSITY COLLEGE, LONDON (UCL) NETS
NORWEGIAN DEFENSE RESEARCH ESTABLISHMENT (NDRE) RING
SRI
ARPANET
SATNET
ROYAL SIGNALS AND RADAR ESTABLISHMENT (RSRE) NET
XEROX
DATA COMMUNICATIONS NETWORK (DCN)
EXPERIMENTAL DATA NETWORK (EDN)
GERMAN DATA COMMUNICATIONS NETWORK (DCN)
ITALIAN DATA COMMUNICATIONS NETWORK (DCN)
SRI
LINCOLN LABORATORIES
WIDEBAND SATELLITE NETWORK
INFORMATION SCIENCES INSTITUTE (ISI)
DEFENSE COMMUNITY ENGINEERING CENTER (DCEC)
ARPANET-TYPE
PACKET RADIO
EXISTING LINE
LOCAL NET
COMMERCIAL NET
FUTURE LINE
SATELLITE NET
GATEWAY

gateways, and between the remote neighbor gateway and the destination host.

Since the X.75 gateway provides virtual-circuit service, it must send messages reliably and in sequence to neighboring X.75 gateways. Flow control between gateways also prevents one gateway from sending more traffic than its neighbors can handle—which is an advantage.

Opponents of the CCITT's virtual-circuit approach to gateways reason that the X.75 architecture is designed to interconnect X.25 networks only and cannot easily link together networks that use different access protocols. These networks include Ethernets, ring networks, and satellite networks. In addition, the X.75 architecture does not provide adaptive routing between networks—when an X.75 call is made, the selection of gateways is fixed. Therefore a failure in one of the X.75 gateways disconnects the call and an alternate route can only be established by making a new call.

The Darpa community has developed an internetwork architecture that allows networks with different access protocols to be interconnected. The Internet System differs in several important ways from the CCITT architecture. For example, gateways are connected directly to networks instead of being connected to other gateways. In addition, traffic is sent across the networks in the form of datagrams instead of via virtual circuits between the networks. And, most importantly, the Darpa Internet uses an adaptive-routing scheme that guarantees that packets exchanged between hosts on different networks travel on the shortest path through gateways. This means that if one gateway fails and there is an alternative gateway available, the alternative gateway will be used automatically without disrupting host-to-host connections.

The current Darpa Internet System consists of more than 20 networks and gateways, several hundred host computers, and several thousand terminals. The Internet networks are connected in a general distributed fashion, with multiple paths between networks and alternate paths that span other networks (Fig. 1). The gateways dynamically decide the best path for a message to be routed to its destination, taking into account topology changes as they occur.

Diverse networks make up the Internet System: terrestrial packet-switching networks such as Arpanet and BBN-Net; satellite networks such as the Atlantic Packet Satellite Network (Satnet) and the Darpa-sponsored Wideband Packet Satellite Network (Wideband); local networks such as Ethernet and the Norwegian Defense Research Establishment (NDRE) Ringnet; and mobile radio networks such as SRI International's packet radio network. These networks vary in characteristics such as message size, speed, delay, reliability, and local address format (Table 1).

It's all in the family

The Darpa research community has developed a family of protocols that provides the mechanisms for host computers to communicate over Internet. These protocols offer services that may be lacking in the underlying networks that make up Internet. As a result of the small number of network requirements, new networks are easily added.

To be part of Internet, a network needs only to be able to deliver messages to a destination and have a minimum message size. The family of Darpa Internet protocols then provides the following services:

- Datagrams
- Addressing
- Message fragmentation and reassembly
- Data reliability
- Message sequencing
- Flow control
- Connections

The Internet System's protocols are a layered family of protocols, as shown in Figure 2. The two main protocols that provide user data transfer are the Internet protocol (IP) [Ref. 1] and the Transmission Control protocol (TCP) [Ref. 2]. In addition, there are protocols for specific applications such as terminal traffic (Telnet), file transfer (FTP), and electronic mail transfer (MTP). Internet also has specialized protocols for functions such as gateway routing, gateway monitoring and control, and error reporting.

Individual network protocols are not specified in the Internet System. Instead, each network has its own access protocols. For instance, Arpanet uses the 1822 Host and IMP protocol (a protocol for interconnection of a host and IMP) [Ref. 3], and Satnet uses the Host Satnet protocol [Ref. 4].

Individual network protocols are used to encapsulate the Internet protocols for transmission across that network. When a message traverses Internet, each gateway creates a new network header appropriate to the next network (Fig. 3).

Datagram delivery

The IP in the second layer of the Internet protocol family transports datagrams across an interconnection of networks. Datagrams are messages that consist of source and destination addresses, plus data. They are not required to be delivered reliably or in sequence. No type of connection needs to be set up to send or receive them. In contrast, virtual-circuit services are provided by high-level end-to-end protocols.

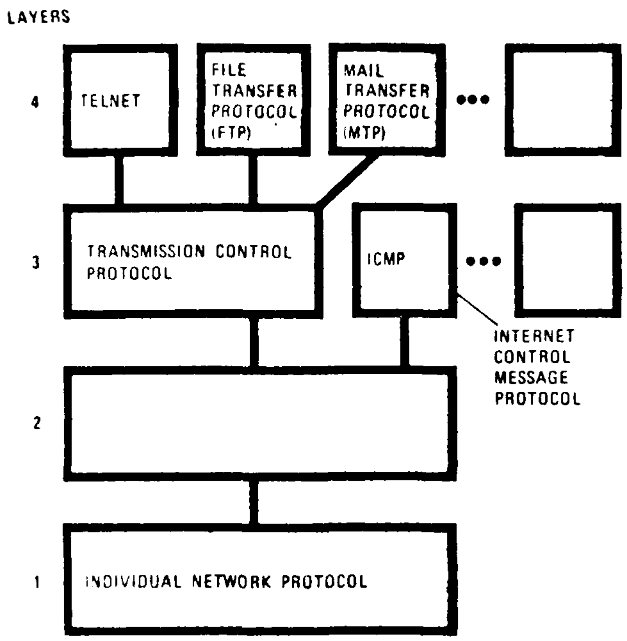
A major advantage of the datagram approach to gateways is that networks are not required to provide many services in order to send a datagram. Therefore, it is comparatively easy to interconnect networks of

Table 1 Network Characteristics

NAME	MESSAGE SIZE IN BYTES	SPEED	DELAY	GUARANTEED DELIVERY	NOTES
ARPANET	1,008	MEDIUM	MEDIUM	YES	
SATNET	256	LOW	HIGH	NO	SATELLITE NETWORK
WIDEBAND	2,000	HIGH	HIGH	NO	SATELLITE NETWORK
PACKET RADIO	254	MEDIUM	MEDIUM	NO	VARYING TOPOLOGY
NDRE RING	2,048	HIGH	LOW	YES	LOCAL NETWORK

WHERE SPEED IS LOW = < 100 KBIT/S, MEDIUM = 100 KBIT/S TO 1 MBIT/S, HIGH = > 1 MBIT/S; AND DELAY IS LOW = < 50 ms, MEDIUM = 50 ms TO 500 ms, OR HIGH = > 500 ms.

2. Internet protocol relationships. *The layered family of Internet protocols permits hosts to communicate over Internet and provides for specific applications.*


LAYERS
4
TELNET
FILE TRANSFER PROTOCOL (FTP)
MAIL TRANSFER PROTOCOL (MTP)
3
TRANSMISSION CONTROL PROTOCOL
ICMP
INTERNET CONTROL MESSAGE PROTOCOL
2
1
INDIVIDUAL NETWORK PROTOCOL

diverse characteristics.

The IP provides two basic services in the second layer: addressing and fragmentation/reassembly. A common address format is maintained across Internet. Addresses are fixed-length (32 bits) and consist of the network number and a local address. The network-number field contains the address of a particular network, and the local-address field contains the address of a host within that network.

The networks that make up Internet have different message sizes. The IP provides a fragmentation/reassembly service to overcome these variations. When a datagram originates in a network that allows large messages, and the datagram must traverse a network with a smaller message limit, the datagram must be broken into smaller "pieces," or fragments. The IP provides a mechanism to permit datagrams to be fragmented and to be later reassembled into one piece at the destination host.

The TCP, in the third layer, is a connection-oriented, reliable end-to-end protocol. It provides the services necessary for reliable message transmission over the Internet System.

The networks that make up Internet are not required to guarantee that all datagrams are delivered. Also, the originator of a datagram does not necessarily know through which networks a datagram will be routed to arrive at its destination. Therefore it is necessary to provide message reliability end-to-end—that is, at the source and the final destination. To address these requirements, the TCP provides reliability, flow control, multiplexing, and connection functions.

Reliability is achieved through checksums (error-detecting codes) and positive acknowledgments of all data. Data that is not acknowledged is retransmitted.

End-to-end flow control lets the receiver of the data regulate the rate at which it is sent. To allow many processes (applications) within a single computer (for example, many terminals talking to one host) to use

3. Message encapsulation. *When a message traverses networks on Internet, individual network protocols are used to encapsulate the Internet protocols for transmission across each network. When the message reaches a gateway, that gateway creates a new network header appropriate to the next network.*

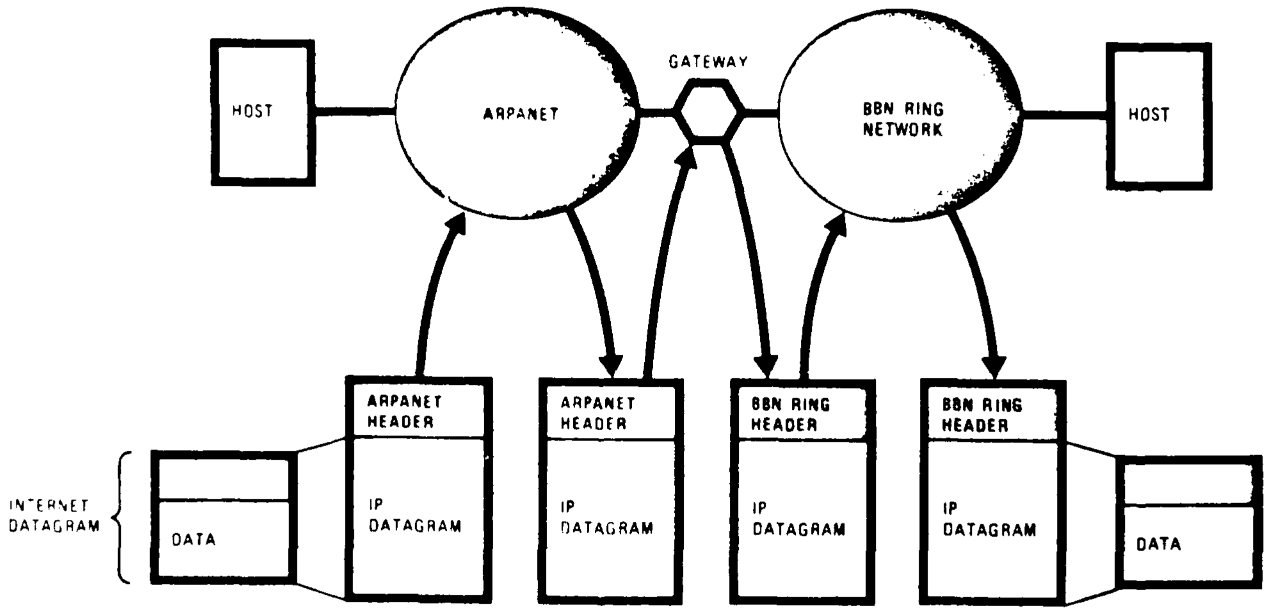


Table 2 Network table for BBN gateway

NETWORK NAME	NET ADDRESS	*ROUTE
SATNET	4	DIRECTLY CONNECTED
ARPANET	10	DIRECTLY CONNECTED
BBN-NET	3	1 HOP VIA RCC 10.3.0.72 (ARPANET 3/72)
PURDUE-COMPUTER SCIENCE	192.5.1	2 HOPS VIA PURDUE 10.2.0.37 (ARPANET 2/37)
INTELPOST	43	2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
DECNET-TEST	38	3 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
WIDEBAND	28	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
BBN-PACKET RADIO	1	2 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
DCN-COMSAT	29	1 HOP VIA MILLS 10.3.0.17 (ARPANET 3/17)
FIBERNET	24	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
BRAGG-PACKET RADIO	9	1 HOP VIA BRAGG 10.0.0.38 (ARPANET 0/38)
CLARK NET	8	2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
LCSNET	18	1 HOP VIA MIT-LCS 10.0.0.77 (ARPANET 0/77)
BBN-TERMINAL CONCENTRATOR	192.1.2	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
BBN-JERICHO	192.1.3	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
UCLNET	11	1 HOP VIA UCL 4.0.0.60 (SATNET 60)
RSRE-NULL	35	1 HOP VIA UCL 4.0.0.60 (SATNET 60)
RSRE-PPSN	25	2 HOPS VIA UCL 4.0.0.60 (SATNET 60)
SAN FRANCISCO-PACKET RADIO-2	6	1 HOP VIA C3PO 10.1.0.51 (ARPANET 1/51)

*NAMES AND ACRONYMS IDENTIFY GATEWAYS IN THE INTERNET SYSTEM.

the protocol simultaneously, the protocol provides for ports to allow individual processes to be identified. The protocol also provides a mechanism for interprocess communications between computers.

Open the gates

A host computer that wants an IP datagram to reach a host on another network must send the datagram to a gateway. A local-network header containing the address of the gateway is attached to the datagram before it is sent into the network. When the packet is received by the gateway, its local-network header is checked for possible errors and the gateway performs any necessary host-to-network protocol functions.

The Internet control message protocol (ICMP) [Ref. 5] is the control protocol associated with the IP that is used to convey error and status information to Internet users. For example, if the header indicates that the packet contains an Internet datagram, then the packet is passed to the Internet header check routine, which performs a number of validity tests on the IP header. Packets that fail these tests are discarded, and an error packet is sent from the gateway to the Internet source of the packet.

After a datagram passes these checks, its Internet destination address is examined to determine if the datagram is addressed to the gateway. Each of the gateway's Internet addresses—one for each network interface—is checked against the destination address in the datagram. If a match is not found, the datagram is passed to the forwarding routine. If the datagram is destined for the gateway, then the datagram is processed according to the protocol in the IP header. Some types of datagrams that might be addressed to the gateway include monitoring packets, gateway routing packets, or remote debugging packets.

Multiple hops

Among other functions, the gateway must make a routing decision for all datagrams that are to be forwarded. The routing procedure provides two pieces of information: which network interface should be used to send the packet, and which destination address should be in the packet's local-network header.

The gateway maintains a network table that contains an entry for each reachable network (Table 2). The entry consists of a network number and either the address of the neighbor gateway on the shortest route to the network or an indication that the gateway is directly connected to the network. A neighbor gateway is one that shares a common network with this gateway. The distance measurement that is used to determine which neighbor is closest is "number of hops." In other words, a gateway is considered to be zero hops from its directly connected networks, one hop from a network that is reachable via one other gateway, and so on. The Gateway-to-Gateway protocol (GGP) [Ref. 6] is used to build the network table.

The gateway tries to match the destination network address in the IP header of the datagram to be forwarded with a network in its network table. If no match is found, the gateway drops the datagram and sends an ICMP packet to the IP source. If the gateway does find an entry for the network in its table, it uses the network address of the neighbor gateway entry as the local network destination address of the datagram. However, if the final destination network is one to which the gateway is directly connected, the destination address in the local-network header is simply built from the destination address in the datagram's IP header.

If the routing procedure decides that an IP datagram is to be sent back out of the same network interface from which it was read, then the source host has chosen a gateway that is not on the shortest path to the IP final destination. The datagram will still be forwarded to the next address chosen by the routing procedure, but a redirect-ICMP packet will also be sent to the IP source host indicating that another gateway should be used to send traffic to the final IP destination.

Break it up

After the routing decision is complete, the datagram is passed to the fragmentation procedure. If the next network through which the datagram must pass has a smaller maximum packet size than the size of the datagram, the gateway will break the datagram into fragments. These fragments are then transported as independent datagrams themselves and are ultimately collected and assembled at the destination host to recreate the original datagram.

The gateway now builds a new network header for the datagram. The gateway uses the information obtained from its routing procedure to choose the proper network interface for the datagram and to build the destination address in the new network header.

The gateway then queues the packet for delivery to its destination. It also enforces a limit on the size of the output queue for each network interface so that a slow network does not unfairly use up all of the gateway's buffers. A packet that cannot be queued because of the limit on the output-queue length is dropped. Whether or not the packet is retransmitted depends on the type of packet.

When the packet finally reaches its destination, the network header is stripped off and the information inside the IP datagram is processed. In addition, if the original datagram was fragmented, the destination host collects all of the fragments and reassembles them into the original datagram.

To provide Internet service, the IP gateway must support a variety of protocols. For example, the gateway has to send and receive packets on its connected network interfaces. Therefore, it must implement all of these networks' access protocols, such as the Arpanet 1822 protocol or the Satnet Host Access protocol.

Since all Internet traffic is sent in the form of Internet datagrams, the gateway must also implement the IP protocol. In addition, the gateway sends control information, such as "This destination network is unreachable," to hosts using the ICMP protocol.

Monitoring and support of gateways is aided by the Cross Network Debugger protocol [Ref. 7], which allows remote debugging of the gateway, and the Host Monitoring protocol [Ref. 8], which allows the gateway to report the status of its interfaces. The gateway also has an internal message generator that is used as a testing facility.

The right way

The IP gateway uses the GGP for four functions concerned with routing:

- Determining if its network interfaces are operational
- Determining if its neighbor gateways are operational
- Building a table of networks that can be reached via neighbor gateways
- Adding new neighbor gateways and new networks to its network table

Gateways use the information obtained from GGP packets to ensure that a datagram uses the best route through Internet to reach its destination.

GGP packets are sent reliably using sequence numbers and an acknowledgment scheme. The gateway determines if its network interfaces are up by sending GGP packets, called "interface probes," addressed to itself every 15 seconds. When a number of these probes have been successfully received, the interface is declared operational. If a number of probes are missed, the interface is declared down.

In order to determine whether other gateways are operating properly, each gateway has a built-in table of neighbor gateways. Every 15 seconds, a gateway will send a GGP echo packet ("neighbor probe") to each of its neighbors to determine which are operational. When a neighbor gateway has echoed a number of probes, it is declared operational. However, if several probes are sent to a neighbor but are not echoed, the neighbor is declared down.

Whenever a gateway determines that there has been a change in Internet routing, such as when it declares one of its network interfaces to be down, it sends a GGP-routing-update packet to each of its neighbors. This packet indicates for each network the distance and address of the gateway on the shortest path to the network.

On receiving a routing update, a gateway will recalculate its network table to ensure that it uses the neighbor on the shortest route to each network. If the routing update packet is from a new neighbor or contains information about a new network, the gateway updates its neighbor or network tables. It thereby learns about new neighbors and networks without having to undergo reconfiguration.

Finding the alternate path

The gateway uses the information in its routing tables to minimize congestion and delay by adapting its routing to the situation. For example, suppose there are two gateways, X and Y, that can be used to reach net-

Table 3 Gateway status report

```
GATEWAY 2  BBN  10.3.0.40  (ARPANET 3/40)   SAT  MAY  8  15:15:06  1982

VERSION   1001

INTERFACES:
          UP:           BBN 4.0.0.61 (SATNET 61)
          UP:           BBN 10.3.0.40 (ARPANET 3/40)

NEIGHBORS:
          UP:           RCC 10.3.0.72 (ARPANET 3/72)
          DOWN:         DCEC 10.3.0.20 (ARPANET 3/20)
          UP:           BRAGG 10.0.0.38 (ARPANET 0/38)
          UP:           C3PO 10.1.0.51 (ARPANET 1/51)
          UP:           R2D2 10.3.0.51 (ARPANET) 3/51)
          DOWN:         NDRE 4.0.0.38 (SATNET 38)
          UP:           UCL 4.0.0.60 (SATNET 60)
          UP:           PTIP 10.2.0.5 (ARPANET 2/5)
          UP:           PURDUE 10.2.0.37 (ARPANET 2/37)
          UP:           MIT-LCS 10.0.0.77 (ARPANET 0/77)
          UP:           MILLS 10.3.0.17 (ARPANET 3/17)
          DOWN:         RING 10.2.0.76 ARPANET 2/76)
          DOWN:         TIU 10.3.0.76 (ARPANET 3/76)

NETWORK TABLE:

NETWORK                      NETWORK ADDRESS      *ROUTE
NAME

 SATNET                          4          DIRECTLY CONNECTED
 ARPANET                        10          DIRECTLY CONNECTED
 BBN-NET                         3          1 HOP VIA RCC 10.3.0.72 (ARPANET 3/72)
 PURDUE-COMPUTER SCIENCE     192.5.1        2 HOPS VIA PURDUE 10.2.0.37 (ARPANET 2/37)
 INTELPOST                      43          2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
 DECNET-TEST                    38          3 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
 WIDEBAND                       28          3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 BBN-PACKET RADIO                1          2 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 SAN FRANCISCO-PACKET RADIO-1    2          UNREACHABLE
 DCN-COMSAT                     29          1 HOP VIA MILLS 10.3.0.17 (ARPANET 3/17)
 FIBERNET                       24          3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 BRAGG-PACKET RADIO              9          1 HOP VIA BRAGG 10.0.0.38 (ARPANET 0/38)
 CLARK NET                       8          2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
 LCSNET                         18          1 HOP VIA MIT-LCS 10.0.0.77 (ARPANET 0/77)
 BBN-TERMINAL CONCENTRATOR   192.1.2        3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 BBN-JERICHO                 192.1.3        3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 UCLNET                         11          1 HOP VIA UCL 4.0.0.60 (SATNET 60)
 RSRE-NULL                      35          1 HOP VIA UCL 4.0.0.60 (SATNET 60)
 BBN-LN-TEST                    41          UNREACHABLE
 RSRE-PPSN                      25          2 HOPS VIA UCL 4.0.0.60 (SATNET 60)
 EDN                            21          UNREACHABLE
 BBN-GT-TEST C               192.0.1        UNREACHABLE
 SAN FRANCISCO-PACKET RADIO-2    6          1 HOP VIA C3PO 10.1.0.51 (ARPANET 1/51)
 BBN-SAT-TEST                   31          UNREACHABLE

                                            *NAMES AND ACRONYMS IDENTIFY GATEWAYS
                                             IN THE INTERNET SYSTEM
```

work A. When gateway X goes down, all of its neighbors will send out routing updates reporting that network A is no longer reachable via gateway X. When a gateway receives this routing update it will recalculate its network table and find that gateway Y can be used to reach network A. Gateways will now forward datagrams through gateway Y to reach network A without disrupting any host-to-host connections.

Putting it together

The IP gateway operates on Digital Equipment Corporation PDP-11 or LSI-11 16-bit processors under a small real-time operating system called the Micro Operating System (MOS), developed by SRI International. MOS provides facilities for multiple processes, interprocess communications, buffer management, asynchronous input/output, and a shareable real-time clock.

There is one MOS network process and accompanying data structure called a netblock, which contains information about, for example, network interface status and queueing for each network that is directly connected to the gateway. Each network process waits for input from one of the gateway's interfaces. When an IP datagram is received, the appropriate network process "wakes up" and calls procedures to forward the datagram toward its destination.

The IP gateway is written in Macro-11 assembly language instead of a higher-level language because memory is limited by the 16-bit address space. The gateway code occupies about 10K words of memory. The MOS operating system occupies an additional 3K words of code space, leaving 15K words for buffers. These buffers are shared by various network processes for reading and writing packets.

Adding support to connect a new network to the IP gateway is a relatively easy task. A programmer must write a device driver that handles the hardware interface of the new network as well as a routine to implement the new host-to-network access protocol. The programmer also creates a gateway-configuration file that contains gateway-specific information, such as interface-device addresses. The macro assembler then assembles a new gateway program. This programming task is simplified because more than 75 percent of the code in all IP gateways is identical because of the modularity of the gateway software.

Keeping order

Fault isolation can be a major problem in the daily operation of a computer network. Some issues that must be resolved are: When communications fails, what is to blame? Is the problem with the host machine, the network, the lines, or the user program?

Internet fault isolation is even more difficult because of the number and diversity of users, networks, paths, and requirements involved. For example, the communications path may traverse many networks and gateways so that the potential sources of communications disruption are multiplied.

The ability to identify areas of congestion is also a more complex task. For example, poor performance can be the result of individual networks failing to provide users with adequate throughput or of a bottleneck in connections between networks.

For Bolt Beranek and Newman, the solution to Internet monitoring and control is to apply techniques much like those used to operate Arpanet. In fact, tools developed by the company to monitor the gateways that are the switching nodes of Internet are similar to those used to monitor the switching nodes in Arpanet.

These tools include a central monitoring facility called the network operational center (NOC) [Ref. 9] that runs on BBN's C/70 computer under the Unix operating system. The NOC regularly receives traffic statistics and reports of important events from each of the Internet gateways. Data communications users can interrogate the NOC to find the current status of any Internet gateway. The monitoring facility then prints a gateway status report (Table 3).

The NOC's status- and event-monitoring capabilities pinpoint hardware and software problems during the operation of Internet. For example, when communications is disrupted between Internet hosts, the NOC monitoring tools help determine whether the problem lies with a gateway, network, communications line, or with one of the Internet hosts. Whenever a gateway receives an erroneous packet, a report that identifies the source of the packet is sent to the NOC. These reports help to diagnose malfunctioning hardware and aid in debugging Internet host software.

References

1. "DOD Standard Internet Protocol," RFC: 791, Information Sciences Institute, University of Southern California, Marina del Rey, Calif., September 1981.
2. "DOD Standard Transmission Control Protocol," RFC: 791, Information Sciences Institute, University of Southern California, Marina del Rey, Calif., September 1981.
3. "Specification for the Interconnection of a Host and IMP," BBN Technical Report 1822, Bolt Beranek and Newman, May 1978.
4. D. McNeill, "Host/Satnet Protocol," IEN: 192, Bolt Beranek and Newman, June 1981.
5. "Internet Control Message Protocol," RFC: 792, Information Sciences Institute, University of Southern California, Marina del Rey, Calif., September 1981.
6. V. Strazisar, "How to Build a Gateway," IEN: 109, Bolt Beranek and Newman, August 1979.
7. J. Haverty, "XNET Formats for Internet Protocol Version 4," IEN: 158, Bolt Beranek and Newman, October 1980.
8. B. Littauer, A. Huang, and R. Hinden, "A Host Monitoring Protocol," IEN: 197, Bolt Beranek and Newman, September 1981.
9. P. Santos, B. Chalstrom, J. Linn, and J. Herman, "Architecture of a Network Monitoring, Control, and Management System," Proceedings of the 5th International Conference on Computer Communications, October, 1980.

Note: References are available from Jake Feinler at the Network Information Center, SRI International, Menlo Park, Calif. ■

===

TCP-IP IMPLEMENTATIONS

June 8, 1982

Source file = [SRI-NIC]<Protocols>TCP-IP-Status.txt

Updates to NIC@SRI-NIC; cc: to POSTEL@ISIF
Technical questions to Postel@ISIF;

===

NOTE

The Network Information Center (NIC@SRI-NIC) has been tasked to maintain the official Internet Gateway Name Table. All internet gateways should be registered with the NIC along with the location of the gateway, names of the connected networks, and a liaison (see examples below). The NIC continues to maintain the ARPANET/DDN Host Name Table as well. To avoid redundancy the NIC, rather than ISI, will now be the focus for updates to this document. The online version will reside in the file, [SRI-NIC]<Protocols>TCP-IP-Status.txt. It may be FTPed using username 'anonymous' and password 'guest' from the SRI-NIC machine (10.0.0.73).

Dr. Jon Postel (Postel@USC-ISIF or (213) 822-1511) is acting as technical coordinator for TCP/IP implementation questions during the transition to the new internet protocols. Specific technical questions about the internet protocols should be directed to him.

I. NETWORK TYPES

The following network types are represented in the internet:

- Packet Switched (ARPANET, DDN, WIN, MINET, EDN)

- Packet Radio (SRI, Ft. Bragg, SAC)

- Packet Satellite (SATNET, WBCNET, MATNET)

- Local Networks (PRONET, ETHERNET (3Mb), ETHERNET (10Mb), BBN-FIBERNET, Ungerman-Bass NET/ONE, LL-LEXNET)

- Public Data Networks (TELENET)

II. ADDRESSES

The internet addresses in this memo are stated as four 8-bit fields with the value of each field given in decimal. See "Assigned Numbers" (RFC-790) for network and protocol number assignments. See "Address Mappings" (RFC-796) and "Host Table Specification" (RFC 810) for a more detailed description of the addressing and naming scheme.

III. INTERNET GATEWAYS

A. GATEWAY TABLE   Date: 29 April 1982

```
BBN-PR-GATEWAY                                   1.0.0.11, 3.0.0.62

   system = MOS
   location = BBN
   networks = BBN-PR, BBN-NET
   liaison = Steve Chipman (Chipman@BBNC)

SRI-R2D2                                        2.0.0.11, 10.3.0.51

   system = MOS
   location = SRI
   networks = SF-PR-1, ARPANET
   liaison = Jim Mathis (Mathis@SRI-KL)

BBN-RING-GATEWAY        3.1.0.11, 31.0.0.61, 41.0.0.5, 192.0.1.5

   system = MOS
   location = BBN
   networks = BBN-NET, BBN-SAT-TEST, BBN-LN-TEST, BBN-TEST-C
   liaison = Alan Sheltzer (Sheltzer@BBN-UNIX)

BBN-TIU-GATEWAY                                  3.2.0.11, 192.0.1.5

   comment = test-gateway
   system = MOS
   location = BBN
   networks = BBN-NET, BBN-TEST-C
   liaison = Alan Sheltzer (Sheltzer@BBN-UNIX)

BBN-PTIP-GATEWAY                                 3.2.0.5, 10.2.0.5

   comment = non-routing
   system = PTIP
   location = BBN
   networks = BBN-NET, ARPANET
   liaison = Steve Chipman (Chipman@BBN)

BBN-FIBRENET-IG                                  3.2.0.50, 24.2.0.1

   system = MOS
   location = BBN
   networks = BBN-NET, BBN-LOCAL
   liaison = Steve Chipman (Chipman@BBN)

BBN-NET-GATEWAY                                  3.3.0.8, 10.3.0.72

   system = MOS
   location = BBN
   networks = BBN-NET, ARPANET
   liaison = Alan Sheltzer (Sheltzer@BBN-UNIX)
```

```
ETAM-IG                                      4.0.0.21, 10.4.0.39

   comment = non-routing
   system = SIMP
   location = ETAM
   networks = SATNET, ARPANET
   liaison = Dale McNeill (McNeill@BBN-UNIX)

NDRE-GATEWAY                        4.0.0.38, 48.0.0.4, 50.0.0.4

   system = MOS
   location = NDRE
   networks = SATNET, NDRE-TIU, NDRE-RING
   liaison = Alan Sheltzer (Sheltzer@BBN-UNIX)

UCL-GATEWAY              4.0.0.60, 11.3.0.42, 32.2.0.42, 35.7.0.0

   system = MOS
   location = UCL
   networks = SATNET, UCLNET, UCL-TAC, RSRE-NULL liaison = Alan
   Sheltzer (Sheltzer@BBN-UNIX)

BBN-GATEWAY                                  4.0.0.61, 10.3.0.40

   system = MOS
   location = BBN
   networks = SATNET, ARPANET
   liaison = Alan Sheltzer (Sheltzer@BBN-UNIX)

SRI-C3PO                                     6.0.0.11, 10.1.0.51

   system = MOS
   location = SRI
   networks = SF-PR-2, ARPANET
   liaison = Jim Mathis (Mathis@SRI-KL)

CLARKSBURG-IG                                8.0.0.30, 10.1.0.71

   comment = non-routing
   system = SIMP
   location = Clarksburg
   networks = CLARKNET, ARPANET
   liaison = Dale McNeill (McNeill@BBN-UNIX)

BRAGG-GW1                                    9.0.0.11, 10.0.0.38

   system = MOS
   location = BRAGG
   networks = BRAGG-PR, ARPANET
   liaison = Ed Perry (Perry@ISID)

MIT-GATEWAY                                  10.0.0.77, 18.8.0.4

   system = MOS
   location = MIT
   networks = ARPANET, LCSNET
   liaison = J Noel Chiappa (JNC@MIT-XX)
```

```
WISC-GATEWAY                                      10.0.0.94, 192.5.2.6

   system = MOS
   location = UWISC
   networks = ARPANET, WISC
   liaison = Rusty Sandberg (rusty@UWISC)

LL-PSAT-IG                                         10.1.0.10, 28.9.0.0

   comment = non-routing
   system = PSAT
   location = LL
   networks = ARPANET, WIDEBAND
   liaison = Walter Milliken (Milliken@BBN-UNIX)

SRI-PSAT-IG                                       10.1.3.51, 28.11.0.0

   comment = non-routing
   system = PSAT
   location = SRI
   networks = ARPANET, WIDEBAND
   liaison = Walter Milliken (Milliken@BBN-UNIX)

PURDUE-CS-GATEWAY                                10.2.0.37, 128.10.0.2

   comment = non-routing
   system = UNIX
   location = PURDUE
   networks = ARPANET, PURDUE-CS
   liaison = Paul McNabb (pam@PURDUE)

DCNET-GATEWAY                                     10.3.0.17, 29.0.1.2

   system = MOS
   location = Linkabit
   networks = ARPANET, DCN-LINKABIT
   liaison = Dave Mills (Mills@ISID)

DCEC-GATEWAY                                      10.3.0.20, 21.0.0.2

   system = MOS
   location = DCEC
   networks = ARPANET, EDN
   liaison = Ed Cain (Cain@EDN-UNIX)

ISI-PSAT-IG                                       10.3.0.22, 28.8.0.0

   comment = non-routing
   system = PSAT
   location = ISI
   networks = ARPANET, WIDEBAND
   liaison = Walter Milliken (Milliken@BBN-UNIX)
```

SAC-GATEWAY 10.3.0.80, 47.0.0.11

system = MOS
location = SAC
networks = ARPANET, SAC-PR
liaison = Ed Perry (Perry@ISID)

DCEC-PSAT-IG 21.0.0.3, 28.10.0.0

comment = non-routing
system = PSAT
location = DCEC
networks = EDN, WIDEBAND
liaison = Walter Milliken (Milliken@BBN-UNIX)

RSRE-GATEWAY 25.6.0.0, 25.13.0.0, 35.6.0.0

system = EMMOS
location = RSRE
networks = RSRE-PPSN, RSRE-NULL
liaison = Andrew Bates (ABates@ISID)

B. BBN MACRO-11 GATEWAYS

Date: 11 May 1982
From: Alan Sheltzer (Sheltzer@BBN-UNIX)

In an effort to provide improved service in the gateways maintained by BBN, a new gateway implementation written in macro-11 instead of BCPL has been developed. The macro-11 gateway provides users with internet service that is functionally equivalent to that provided by the current BCPL gateways with the following exceptions:

1. Packets with options will be fragmented if necessary.

2. ICMP protocol is supported. The gateway sends Time Exceeded, Parameter Problem, Echo, Information Request, Destination Unreachable, and Redirect ICMP messages.

3. Initially, Source Quench and Timestamp packets will not be supported.

4. Class A, B, and C Network Address formats as specified in the September 1981 Internet Protocol Specification (RFC 791) are supported.

5. The gateway contains an internetwork debugger (XNET) that allows the gateway to be examined while it is running.

6. Buffer space is greatly expanded to provide better throughput.

ARPANET RFNMs are counted so the gateway will not send more than 8 outstanding messages to an ARPANET host.

The following gateways are now using this implementation:

BBN-PR-GATEWAY
BBN-RING-GATEWAY
BBN-TIU-GATEWAY
BBN-NET-GATEWAY
NDRE-GATEWAY
UCL-GATEWAY
BBN-GATEWAY
SRI-C3PO
DCEC-GATEWAY

IV. HOST IMPLEMENTATIONS OF TCP/IP

1. BBN H316 and C/30 TAC

Date: 18 November 1981
From: Bob Hinden (Hinden@BBN-UNIX)

The Terminal Access Controller (TAC) is a user Telnet host that supports TCP/IP and NCP host-to-host protocols. It runs in 32K H-316 and 64K C/30 computers. It supports up to 63 terminal ports, and connects to a network via an 1822 host interface.

The TAC TCP/IP conforms with RFC-791 and RFC-793 specifications with the following exceptions:

1. IP options are accepted but ignored.
2. All TCP options except maximun segment size are not accepted.
3. Precedence, security, etc. are ignored.

The TAC also supports Packet core, TAC Monitoring, Internet Control Message Protocol (ICMP), and a subset of the Gateway-Gateway protocols.

For more information on the TAC's design, see IEN-166.

All major features have been implemented except Class B and C addressing, IP reassembly, and TCP Urgent handling. These will be done in the near future.

Hosts:

TAC	ADDRESS	Date: 12 May 1982
ABER	10.2.0.29	
AFGL	10.2.0.66	
AFSD	10.1.0.65	
ANDRWS	10.1.0.67	
BBN	1.0.0.83	
BBNCC1	3.1.0.3	
BBNCC2	3.2.0.3	
BRAGG	10.2.0.38	
CCA	10.2.0.31	
CHINA	10.2.0.85	

DARCOM	10.2.0.50
DAVID	10.2.0.81
DEP56	3.0.0.10
DIV-5	3.2.0.9
GUNTER	10.2.0.13
NBS	10.2.0.19
NSWC	10.2.0.84
PAXRV	10.3.0.97
SAC	10.2.0.80
STLA	10.2.0.61
UCL	11.2.0.42
USGS2	10.1.0.69
USGS3	10.1.0.70
WASH	10.2.0.91
YUMA	10.2.0.75

2. BBN TENEX and TOPS20

Date: 23 November 1981
From: Charles Lynn (CLynn@BBNA)

TCP and IP are available for use with the TENEX operating system running on a Digital KA10 processor with BBN pager. TCP and IP are also available as part of TOPS20 Release 3A and Release 4 for the Digital KL10 and KL20 processors.

Above the IP layer, there are two Internet protocols within the monitor itself (TCP and GGP). In addition, up to eight (actually a monitor assembly parameter) protocols may be implemented by user-mode programs via the "Internet User Queue" interface. The GGP or Gateway-Gateway Protocol is used to receive advice from Internet Gateways in order to control message flow. The GGP code is in the process of being changed and the ICMP protocol is being added.

TCP is the other monitor-supplied protocol, and it has two types of connections -- normal data connections and "TCP Virtual Terminal" (TVT) connections. The former are used for bulk data transfers while the latter provide terminal access for remote terminals.

Note that TVTs use the standard ("New") TELNET protocol. This is identical to that used on the ARPANET with NCP and in fact, is largely implemented by the same code.

At the IP level, fragmentation and reassembly are currently being tested. The Security option can be parsed, but no code for doing preemption of resources has been writen. Certain other security-related features are implemented.

Performance improvements, support for the new address formats, and User and Server FTP processes above the TCP layer are under development.

The most recent release of IP and TCP TOPS20 software is based on the Multinet network interface and was made about 15 May 82.

Hosts:

BBNA 10.3.0.5 Date: 27 May 1981

Contact: Steve Chipman (617)491-1850 Chipman@BBNC
TCP Services:

Port	Service
23	Telnet

Test Account: None

BBNB 10.0.0.49 Date: 27 May 1981

Contact: Steve Chipman (617)491-1850 Chipman@BBNC
TCP Services:

Port	Service
1	Telnet

Test Account: None

BBNC 10.3.0.49 Date: 27 May 1981

Contact: Steve Chipman (617)491-1850 Chipman@BBNC
TCP Services:

Port	Service
23	Telnet

Test Account: None

BBND 10.1.0.49 Date: 27 May 1981

Contact: Steve Chipman (617)491-1850 Chipman@BBNC
TCP Services:

Port	Service
23	Telnet

Test Account: None

BBNE 10.0.0.5 Date: 27 May 1981

Contact: Steve Chipman (617)491-1850 Chipman@BBNC
TCP Services: None
Test Account: None

BBNF 3.2.0.51 Date: 27 May 1981

Contact: Steve Chipman (617)491-1850 Chipman@BBNC
TCP Services:

Port	Service
23	Telnet

Test Account: None

```
BBNG      10.1.0.51     Date: 27 May 1981

   Contact:   Steve Chipman        (617)491-1850     Chipman@BBNC
   TCP Services:
      Port   Service
      ----   -------
       23    Telnet
   Test Account: None

ISIB      10.3.0.52     Date: 27 May 1981

   Contact:   Dennis Smith         (213)822-1511     Smith@ISIB
   TCP Services:
      Port   Service
      ----   -------
       23    Telnet
   Test Account: None

ISIC      10.2.0.22     Date: 27 May 1981

   Contact:   Dennis Smith         (213)822-1511     Smith@ISIB
   TCP Services:
      Port   Service
      ----   -------
       23    Telnet
   Test Account: None

ISID      10.0.0.27     Date: 27 May 1981

   Contact:   Dennis Smith         (213)822-1511     Smith@ISIB
   TCP Services:
      Port   Service
      ----   -------
       23    Telnet
       57    MTP Mail
   Test Account: None

ISIE      10.1.0.52     Date: 27 May 1981

   Contact:   Dennis Smith         (213)822-1511     Smith@ISIB
   TCP Services:
      Port   Service
      ----   -------
       23    Telnet
       57    MTP Mail
   Test Account: None

ISIF      10.2.0.52     Date: 27 May 1981

   Contact:   Dennis Smith         (213)822-1511     Smith@ISIB
   TCP Services:
      Port   Service
      ----   -------
       23    Telnet
       25    SMTP Mail
       57    MTP Mail
   Test Account: TCP-TEST
```

3. DEC TOPS20

Date: 3 May 1982
From: Kevin Paetzold (Paetzold@DEC-MARLBORO)

DEC is merging the BBN TCP/IP software into TOPS-20, and implementing a different JSYS interface that is more consistent with the other TOPS20 I/O JSYSs.

Hosts: none indicated

4. SRI TENEX (AUGUST and FOONEX)

Date: 1 July 1982
From: Henry Miller (MILLER@SRI-NIC)

SRI has implemented TCP and IP for the TENEX (FOONEX and AUGUST) operating system running on F3 and F4 Foonly processors. The SRI TENEX TCP/IP is written in MACRO and resides in the operating system. It was adapted from the BBN and ISI versions of TENEX TCP/IP. The work was supported by the DCA/NIC contract, and the NIC version of TENEX TCP/IP is available for other machines running TENEX, FOONEX, or AUGUST.

Hosts:

SRI-NIC 10.0.0.73 Date: 1 July 1982

Contact: Henry Miller (415)859-5303

TCP Services

Port	Service
23	Telnet
37	Time (currently not running)
42	Name
43	Whois
101	Host

UDP Services

Port	Service
101	Host

SRI-CSL 10.2.0.2 Date: 8 May 1982

Contact: Geoff Goodfellow (415)859-3098 GEOFF@SRI-CSL

OFFICE machines

Contact: Steve Kudlak (408)446-6102 KUDLAK@OFFICE

5. TOPS10

Date: 1 July 1982
From: Don Provan (Don.Provan@CMU-10A)

Don Provan (Don.Provan@CMU-10A) is currently implementing a TOPS10 version of TCP/IP under contract to the Air Force.

6. BBN UNIX 11/70

Date: 14 May 1981
From: Jack Haverty (haverty@BBN-UNIX)

This TCP implementation was written in C. It runs as a user process in version 6 UNIX, with modifications added by BBN for network access. It does not perform reassembly, and has no separate IP user interface. It supports user and server Telnet.

1. Hardware - PDP-11 running UNIX version 6, with BBN IPC additions.

2. Software - written in C, requiring 22K instruction space, 15K data space. Supports 10 connections.

3. Status - TCP has been essentially completed since March, 1979, and no additional work has been done on it since then.

4. Unimplemented protocol features

   A. TCP - Ignores options except S/P/T.
   Discards out-of-order segments.

   B. IP - Does not support fragmentation or reassembly.
   Ignores options.

5. Documentation - "TCP/PSIP Development Report", and "TCP Software Documentation", both BBN reports.

This implementation was done under contract to DCEC. It is installed currently on several PDP-11/70s and PDP-11/44s. Contact Ed Cain at DCEC <Cain@EDN-UNIX> for details of further development.

Hosts:

```
    BBN-UNIX         10.0.0.63  Date: 27 May 1981

       Contact:  Tom Blumer         (617)491-1850     tpb@BBN-Unix
       TCP Services:
          Port   Service
          ----   -------
          23    Telnet
       Test Account: TCP-TEST
```

7. DCEC PDP-11 UNIX

Date: 29 January 1982
From: Ed Cain <cain@EDN-UNIX>

This TCP/IP/ICMP implementation runs as a user process in version 6 UNIX, with modifications obtained from BBN for network access. IP reassembles fragments into datagrams, but has no separate IP user interface. TCP supports user and server Telnet, echo, discard, internet mail, and a file transfer service. ICMP generates replies to Echo Requests, and sends Source-Quench when reassembly buffers are full.

1. Hardware - PDP-11/70 and PDP-11/45 running UNIX version 6, with BBN IPC additions.

2. Software - written in C, requiring 25K instruction space, 20K data space. Supports 10 connections.

3. Unimplemented protocol features:

   A. TCP - Ignores all options (work in progress to implement the max-seg-size option (the only defined option)). Discards out-of-order segments (work in progress to utilize out-of-order segments).

   B. IP - Ignores options except Security/TCC

Hosts:

EDN-HOST1 21.1.0.1 Date: 27 May 1981

Contact: Ed Cain (703)437-2578 Cain@EDN-Unix

TCP Services:

Port	Service
1	THP
7	Echo
9	Discard
17	Short Text
23	Telnet
25	SMTP Mail
65	List of services
53	AUTODIN II FTP
57	MTP Mail
79	Finger: UNIX "dpy" command.

Test-Account: TCP-TEST

EDN-HOST3 21.0.0.3 Date: 27 May 1981

Contact: Ed Cain (703)437-2578 Cain@EDN-Unix

TCP Services:

Port	Service
1	THP
7	Echo
9	Discard
17	Short Text
23	Telnet
25	SMTP Mail
65	List of services
53	AUTODIN II FTP
57	MTP Mail
79	Finger: UNIX "dpy" command.

Test-Account: TCP-TEST

EDN-UNIX 10.3.0.20 Date: 27 May 1981

Contact: Ed Cain (703)437-2578 Cain@EDN-Unix

TCP Services:

Port	Service
1	THP
7	Echo
9	Discard
17	Short Text
23	Telnet
25	SMTP Mail
65	List of services
53	AUTODIN II FTP
57	MTP Mail
79	Finger: UNIX "dpy" command.

Test-Account: TCP-TEST

8. BBN UNIX C70

Date: 18 November 1981
From: Rob Gurwitz (Gurwitz@BBN-UNIX)

The C/70 processor is a BBN-designed system with a native instruction set oriented toward executing the C language. It supports UNIX Version 7 and provides for user processes with a 20-bit address space. The TCP/IP implementation for the C/70 was ported from the BBN VAX TCP/IP, and shares all of its features.

This version of TCP/IP is running experimentally at BBN, but is still under development. Performance tuning is underway, to make it more compatible with the C/70's memory management system.

Hosts:

BBNT 3.3.0.7 Date: 27 January 1982

9. BBN UNIX VAX

Date: 27 May 1981
From: Judy Gordon (JGordon@BBN-UNIX)

BBN has developed an implementation of TCP/IP for DEC's VAX(TM) family of processors, that runs under the Berkeley 4.1BSD version of UNIX(TM). The development effort was funded by DARPA.

Some important features of the BBN VAX TCP/IP are that it runs in the UNIX kernel for enhanced performance, it is a complete implementation of the TCP and IP protocols, and provides facilities for direct user access to the IP and underlying network protocols. The IP module supports checksums, option interpretation, fragmentation and reassembly, extended internet address support, gateway communication with ICMP, and support of multi-homing (multiple interfaces and addresses on the same or different networks). The TCP supports checksums, sequencing, the ability to pass options through to the IP level, and advanced windowing and adaptive retransmission algorithms. Support is also provided for the User Datagram Protocol (UDP).

In addition to the TCP/IP software for the VAX, BBN has developed implementations of the TELNET Virtual Terminal Protocol, File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP), for use with TCP. These protocols are operated as user level programs. Also provided are network programming support tools, such as network name/address manipulation libraries, status, tracing, and debugging tools.

The TCP/IP and higher level protocol software are now available direct from BBN. The software is distributed on a 1600 bpi tar format tape, containing the sources and binaries for a 4.1BSD UNIX kernel containing the network modifications and the sources and binaries for the higher level protocols and support software. Documentation is provided in the form of a set of UNIX manual pages for the network access device, user programs, and libraries. In addition, a detailed installation document is provided. Device drivers are supplied for the ACC LH/DH-11 IMP interface and the Proteon Assoc. PRONET Local Network Interface.

The tape is available for a $300.00 duplication fee to Berkeley 4.1BSD licensees. To order the tape, contact:

Ms. Judy Gordon
Bolt Beranek and Newman, Inc.
10 Moulton St.
Cambridge, MA 02238
617-497-3827
jgordon@bbn-unix

You will then receive a copy of the licensing agreement. Tapes will be mailed upon receipt of a completed agreeement and the distribution fee.

This tape is supplied as-is to 4.1BSD licensees, with no warranties or support expressed or implied. BBN would be pleased to arrange separate agreements for providing installation assistance and/or software support services, if desired.

UNIX is a trademark of Bell Laboratories. VAX is a trademark of Digital Equipment Corporation.

Hosts:

```
   BBN-VAX         10.2.0.82  Date: 27 May 1981

      Contact:   Rob Gurwitz  (617)491-1850   Gurwitz@BBN-Unix
      TCP Services
         Port   Service
         ----   -------
          23    Telnet
      Test Account: TCP-TEST
```

10. ISI UNIX VAX

Date: 8 June 1982
From: Dennis Smith (Smith@ISIB)

Hosts:

```
   ISI-VAXA   10.2.0.27   Date: 8 June 1982

      Contact:   Dennis Smith       (213)822-1511     Smith@ISIB
      TCP Services:
         Port   Service
         ----   -------
          23    Telnet
      Test Account: TCP-TEST
```

11. DTI VMS VAX

Date: 12 Mar 1982
From: John Schur (schur at dti-vms)

This TCP implementation is written in C for the VMS operating system. It uses ACP's for the TCP and IP processes, and supports user level interfaces to these ACP's.

The implementation fully conforms to the TCP (RFC 793), IP (RFC 791) and ICMP (RFC 792) specifications. Higher level protocol services include user and server TELNET, FTP, and SMTP.

1. Hardware - VAX 11/780 or 11/750 running VMS 2.2 or later, and ACC LH/DH-11 interface (other devices will be supported in future according to user interest).

2. Software - written in mostly C and some MACRO. Supports a user-definable number of connections.

3. Status - TCP/IP ACP's are currently in testing stages, with field test sites to begin use in April.

4. Protocol Features Supported:

IP:

Fragmentation/Reassembly: reassembly is supported, but fragmentation is not implemented.

Options: all options are generated and interpreted.

Reassembly timeout: fixed value. Oldest fragments are discarded first when buffers fill up.

TCP:

Options: All defined options are implemented.

Urgent, Push: Supported as per specifications.

Retransmission: Timeouts employ exponential backoff until a limit is reached, at which time user is notified.

Window strategy: Window size is larger than the actual available buffer space by the maximum size of an internal buffer.

Please contact DTI for further information.

Hosts: None indicated

12. SRI LSI-11

Date: 15 May 1981
From: Jim Mathis (Mathis.tscb@SRI-UNIX)

The IP/TCP implementation for the Packet Radio terminal interface unit is intended to run on an LSI-11 under the MOS real-time operating system. The TCP is written in MACRO-11 assembler language. The IP is currently written in assembler language; but is being converted into C. There are no plans to convert the TCP from assembler into C.

The TCP implements the full specification, although the current user interface lacks a mechanism to communicate URGENT pointer information between the TCP and the higher-level software. The code for rubber-EOL has been removed in anticipation of a change to the specification. The TCP appears to be functionally compatible with all other major implementations. In particular, it is used on a daily basis to provide communications between users on the Ft. Bragg PRNET and ISID on the ARPANET.

The IP implementation is reasonably complete, providing fragmentation and reassembly; routing to the first gateway; and a complete host-side GGP process. Currently the source quench message is ignored. No IP options are generated and all received options are ignored.

A measurement collection mechanism is currently under development to collect TCP and IP statistics and deliver them to a measurement host for data reduction.

Hosts: None indicated

13. BBN HP-3000

Date: 14 May 1981
From: Jack Sax (sax@BBN-UNIX)

The HP3000 TCP code is in its final testing stages. The code includes under the MPE IV operating system as a special high priority process. It is not a part of the operating system kernel because MPE IV has no kernel. The protocol process includes TCP, IP, 1822 and a new protocol called HDH which allows 1822 messages to be sent over HDLC links. The protocol process has about 8k bytes of code and at least 20k bytes of data depending on the number of buffers allocated.

The TCP code is believed to support all features except rubber EOL. The IP code currently supports fragment reassembly but not fragmentation. In addition provisions have been made to allow the IP layer to accept and act on routing and source quench messages. These features will be added sometime this summer. Security and precedence are currently ignored.

In addition to TCP, the HP3000 has user and server TELNET as well as user FTP. A server FTP may be added later.

A complete description of the implementation software can be found in IEN 167.

For further information see BBN Report 4856 (January 1982).

Hosts: none indicated

13. MIT MULTICS

Date: 29 December 1981
From: Dave Clark <Clark@MIT-Multics>
Michael Greenwald <Greenwald@MIT-Multics>

Multics TCP/IP is implemented in PL/1 for the HISI 68/80. It has been in experimental operation for about 2 years; it can be distributed informally as soon as certain modifications to the system are released by Honeywell. The TCP and IP package are currently being tuned for performance, especially high throughput data transfer.

It is believed that the implementation fully conforms to the DOD standard. It also supports most relevant features of GGP and ICMP, ıncluding redirect packets. The IP layer is a gateway, and supports fragmentation as well as reassembly.

We don't do much with options. The only exception to this is TCP max segment size - with which you can coax us to send you TCP monster packets. (The record so far is 5000 octet packets between mit-multics and cisl-multics [with a clone of our code], but that was only for testing purposes.)

Higher level services include user and server telnet, and a full function MTP mail forwarding package. We also have a preliminary SMTP implementation.

The TCP and IP contain good logging and debugging facilities, which have proved useful in the checkout of other implementations. Please contact us for further information.

Hosts:

```
   MIT-Multics      10.0.0.6   Date: 27 December 1981

      Contact:  Dave Clark (617)253-6003  Clark@MIT-Multics
                Mike Greenwald (617)253-6042  Greenwald@MIT-Multics
```

TCP Services:

Port	Service
7	Echo
9	Discard
23	Telnet
25	SMTP
37	Special MIT Time Server
57	MTP Mail

UDP Services

Port	Service
14	Name Server
69	TFTP

Test Account: TCP_Test

MIT-DevMultics 10.3.0.31 Date: 27 December 1981

Contact: Dave Clark (617)253-6003 Clark@MIT-Multics
Michael Greenwald (617)253-6042 Greenwald@MIT-Multics

TCP Services:

Port	Service
1	Old-Telnet
7	Echo
9	Discard
23	Telnet
37	Special MIT Time Server

UDP Services

Port	Service
14	Name Server
69	TFTP

Test Account: None

14. UCLA IBM

Date: 18 Jan 1982
From: Bob Braden (Braden@ISI)

Implementation Status -- IP/TCP for IBM 360/370 under OS/MVS or OS/MVT.

1. Hardware

   IBM 360 or 370 CPU. IMP connected to Basic Multiplexor channel using ACC interface box.

2. Operating System

   OS/MVS with ACF/VTAM. An OS/MVT version is also available.

Installation of the MVT version includes a number of operating system extensions and modifications; the MVS version uses an unmodified IBM system.

The ARPANET control program operates as a user job, which must be declared non-swappable to MVS and occupy a high performance group. Under MVT, it must have high dispatching priority.

3. Implementation Language

   Assembler H.

4. Protocol features supported:

   A. IP PROTOCOL:

      (1) Fragmentation/reassembly: performs reassembly. Does not fragment, assuming that higher-level protocol (TCP) will create suitable size segments during packetizing.

      (2) Options: all internet options accepted but ignored. None are sent (in particular, no error options).

      (3) Identifier selection: uses globally-unique identifiers for transmitted segments, independent of destination.

      (4) Reassembly timeout: fixed value (30-60 seconds), independent of time-to-live field. Packets are discarded if time-to-live field is zero.

      (5) Gateway functions: Fixed routing, based either on its own host table (for locally-initiated association) or on gateway from which first packet received (for remotely-initiated association). Currently unable to select an alternate gateway if the original choice fails.

      (6) ICMP: Accepts GGP, has not yet been converted to ICMP.

      (7) Type of Service: default Type of Service set, may cause either Subtype 0 or Subtype 3 (Uncontrolled) packets to be sent.

   B. TCP PROTOCOL:

      (1) Precedence, security fields: not set or tested.

      (2) TCP Options: no options generated. All options accepted but ignored.

      (3) Urgent: may be sent and received by user process.

      (4) EOL: may be sent by user process, but received EOL's are not passed to user process because input uses a circular buffer.

(6) Retransmission: successive retransmissions use exponential backoff. Base time is 2 times observed exponentially weighted round-trip time. Round-trip time is measured as initial packet transmission to complete acknowledgment. Retransmits slowly into zero window.

(7) Initial Sequence Number: derived from system clock.

(8) Window strategy: uses conservative strategy, never advertising a receive window larger than the space available in the circular buffer.

(9) ACK generation: always sends <ACK> in response to receipt of a non-empty packet. As user process removes bytes from buffer, optimizing algorithm determines when to generate <ACK> to inform sender of larger window.

5. UDP

UDP has not yet been implemented.

6. User-Level Protocols Available with TCP

User and Server Telnet.

FTP has not yet been converted to use TCP.

Hosts

UCLA-CCN 3033   10.1.0.1 Date: 27 May 1981

Contact: Bob Braden   (213)825-7518   Braden@ISI
TCP Services:

Port	Service
7	Echo
23	Telnet (TSO & NETSTAT)

Test Account: None

15. LINKABIT DCNET Internet Software

Date: 17 April 1982
From: Dave Mills (Mills at ISID)

The DCNET internet software system has been developed with DARPA sponsorship over the last three years and used extensively for testing, evaluation and experimentation with other implementations. It currently runs in a sizable number of PDP11s and LSI-11s with varying configurations and applications. The system is designed to be used with the DCNET local network and BOS/VOS operating system for a multi-media internet workstation (so-called "fuzzball"), which operates using emulation techniques to support ordinary RT-11 system and application programs.

However, the system has also been used on other networks, including ARPANET, and with other operating systems, including RSX-11. An RSX-11 based version is presently used to support the INTELPOST electronic-mail network.

The DCNET system consists of a package of MACRO-11 and C modules structured into levels corresponding to local-net, IP, TCP and application levels, with user interfaces at each level. The local-net level supports several comunication devices, including synchronous and asynchronous serial lines, 16-bit parallel links and 1822 interfaces. Hosts using these devices have been connected to ARPANET IMPs, Satellite IMPs, BCPL and MACRO-11 Internet Gateways, SRI Port Expanders and to the DCNET local network. When used on DCNET the system provides automatic routing, time-synchronization and error-reporting functions.

The IP level conforms to the RFC-791 specification, including fragmentation, reassembly, extended addressing and options, but currently does not interpret options. A full set of ICMP features compatible with RFC-792 is available, including destination-unreachable, timestamp, redirect and source-quench messages. Destination-unreachable and source-quench information is conveyed to the user level via the TCP and raw-datagram protocol modules. Internet gateway (routing and non-routing) facilities compatible with IEN-109 (as amended) can be included on an optional basis. This support can be configured to include hierarchically structured gateways and subnets.

The TCP level conforms to the RFC-793 specification, including PUSH, URGENT and options, but currently does not interpret options. Its structure is based on circular buffers for reassembly and retransmission, with repacketizing on each retransmission. Retransmission timeouts are dynamically determined using measured roundtrip delays, as adjusted for backoff. Data flow into the network is controlled by measured network bandwidth, as adjusted by source-quench information. Features are included to avoid excessive segment fragmentation and retransmission into zero windows. The user interface level provides error and URGENT notification, as well as a means to set outgoing IP/TCP options.

A raw-datagram interface is available for XNET (IEN-158), UDP (RFC-768) and similar protocols. It includes internal congestion and fairness controls, multiple-connection management and timestamping. Protocols above UDP supported in the present system include Time Server (IEN-142) and Name Server (IEN-116). A number of user-level protocol modules above TCP have been built and tested with other internet hosts, including user/server TELNET (RFC-764) user/server FTP (RFC-765), user/server MTP (RFC-780), user/server SMTP (RFC-788) and various other file-transfer, debugging and control/monitoring protocols.

Code sizes and speeds depend greatly on the system configuration and features selected. A typical 30K-word LSI-11/2 single-user configuration with all features selected and including the operating system, device drivers and all buffers and control blocks, leaves about 16K words for user-level application programs and protocol modules. A typical 124K-word LSI-11/23 configuration provides the same service to a half-dozen individually relocated users. Disk-to-disk FTP transfers across a DMA interprocessor link between LSI-11/23s operate in the range 20-30 Kbps with 576-octet packets. The 124K-word PDP11/34 INTELPOST adaptation supports two 56-Kbps lines and a number of lower-speed lines.

DCNET Supported Protocols

All DCNET hosts can support the following protocols:

Number	Name	Protocol
1	ICMP	Internet Control Message Protocol
3	GGP	Gateway-Gateway Protocol
4	GMP	Gateway Monitoring Protocol
6	TCP	Transmission Control Protocol
7	UCLP	University College London Protocol
15	XNP	XNET Cross-Net Debugger Protocol
17	UDP	User Datagram Protocol
19	DCNP	DCNET Protocol
63	LNP	DCNET HELLO Protocol
71	SMP	SIMP Monitoring Protocol

Notes:
1. XNP datagrams directed to some DCNET hosts will simulate power-up reset followed by entry to the downline loader firmware.

2. GMP, UCLP, XNP, UDP, DCNP and SMP protocols are supported only when the relevant protocol modules are active.

DCNET Supported Services

All DCNET hosts can support the following services and associated port numbers:

Port	Name	Service
7	ECHO	Echo server
9	SINK	Sink (discard) server
19	TTYTST	Traffic generator server
21	FTP	File transfer (FTP) server
23	TELNET	Virtual terminal (TELNET) server
25	SMTP	Simple internet mail (SMTP) server
37	TIME	Time server
42	NAME	Name server
45	MPM	Internet message (MPM) server
47	NIFTP	File transfer (NIFTP) server
57	MTP	Internet mail (MTP) server
87	TALK	Operator Intercom server

Notes:

1. All servers operate with TCP. The ECHO, SINK, TIME and NAME servers operate with UDP as well.

2. The FTP, NAME, NIFTP, MTP and SMTP servers require a disk. Most TELNET server funcions require a disk.

3. The ECHO server forwards datagrams after interchanging addresses and ports. The SINK server discards TCP data at the user level. The TTYTST server repeats a test message continuously until the connection is closed by the user.

4. The FTP server is compatible with the minimal implementation of RFC-765.

5. The TELNET server is compatible with RFC-764, including IP/URGENT, but excluding negotiations, and requires "local echo." It supports an RT-11 emulator which can run most utilities and user programs for that system.

6. The TIME server is compatible with IEN-142. It is synchronized to the DCNET time standard, which provides accurate timekeeping to within a few milliseconds relative to NBS radio time broadcasts.

7. The NAME server is compatible with IEN-116. The host name-address tables contain all the ARPANET hosts from the NIC database, together with all DCNET and many other internet hosts.

8. The NIFTP and MPM servers are presently but empty shells.

9. The MTP server is compatible with the minimal implementation of RFC-780 and requires "recipients first."

10. The SMTP server is compatible with the minimal implementation of RFC-788.

11. The TALK server links the operator terminal to the TELNET connection in full-duplex mode.

==

TCP-IP DIGEST

Contributions to Mike Muuss, Coordinator (Mike@BRL)
Back issues in directory [SRI-NIC]<TCP-DIGEST>

==

Michael Muuss at the Ballistics Research Laboratory is coordinating an informal online special-interest digest called TCP-IP DIGEST. Those interested in general news items concerning the TCP/IP protocols are encouraged to add their names to the distribution list by sending a network message to

TCP-REQUEST@BRL or TCP-IP-REQUEST@BRL

Implementors are also encouraged to contribute news items about their versions of TCP/IP, or anything else of related interest to

TCP-IP@BRL

Back issues are available online only from the directory <TCP-DIGEST> on the SRI-NIC machine (10.0.0.73). The issues may be FTPed from your local host using username 'anonymous', password=guest. The files are in the form

```
<TCP-DIGEST>TCP-V1N01.txt
<TCP-DIGEST>TCP-V2N09.txt
<TCP-DIGEST>TCP-V3N10.txt
etc.
```

The newsletter is the place to report milestones, ask questions of fellow programmers, discuss interesting technical issues, and generally keep in touch. It does not have a regular publishing cycle, and is published whenever Mike has enough material of interest and enough time to get it organized and sent.